GUIDE DU TO 7

Cet ouvrage a été réalisé sous la direction de Benoît de MERLY.

Nous remercions THOMSON pour ses encouragements à la rédaction de ce guide ainsi que pour la mise à disposition des photographies qui illustrent le premier chapitre.

Première édition

Tome 1 ISBN : 2-904457-04-6

EDIMICRO

DÉPARTEMENT ÉDITIONS DE F.D.S. SARL

121-127, avenue d'Italie, 75013 Paris

GUIDE DU TO 7

par

Jean-François BIEBER
Alain PERBOST
Gilles RENUCCI



Sommaire

ANNEXES

CHAPITRE **1**

Présentation générale du TO 7

1.1 INTRODUCTION

Le Thomson TO7 est un ordinateur. Il est habituellement désigné par le terme micro-ordinateur, car il est extrêmement petit par rapport aux premiers ordinateurs. Son "coeur" est un microprocesseur. Comme vous pouvez le voir sur la photo A, le TO7 a un clavier qui est utilisé pour "entrer" des données. A l'intérieur, sont placés de nombreux circuits intégrés parmi lesquels figurent le microprocesseur, les mémoires (voir §1.7.). Au début, vous n'avez nullement besoin d'avoir la moindre connaissance en électronique pour utiliser votre TO7.

A côté du clavier sont situés le crayon lumineux dans un logement spécial (voir §1.4) et le lecteur de cartouches préenregistrées en mémoire morte MEMO 7 (voir §1.5).

La sortie vidéo est présente à l'arrière de l'appareil et délivre des signaux RVB par l'intermédiaire d'une prise Péritel qui est installée sur tous les téléviseurs couleur produits depuis 1980 (voir §1.3).

La photo A montre un TO7 muni d'une télévision couleur.

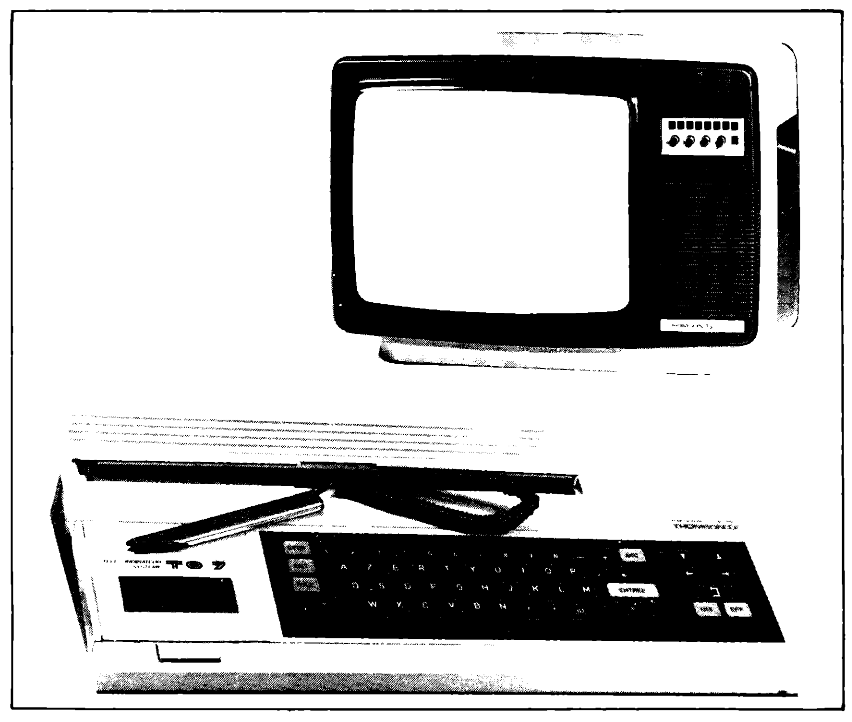


PHOTO A : LE SYSTEME TO7

Votre micro-ordinateur TO7 offre de nombreuses possibilités graphiques à ses utilisateurs. Vous pourrez ainsi utiliser des jeux graphiques haute-résolution, tracer des courbes sur l'écran ou dessiner à l'aide du crayon lumineux (voir §1.4).

Avec le TO7, vous pourrez vous contenter d'utiliser des programmes tout faits (jeux, gestion familiale...), mais vous pourrez aussi faire vos programmes, puis utiliser des disquettes, communiquer avec d'autres ordinateurs (voir §1.7). Bientôt, votre TO7 s'avèrera indispensable.

1.2 LE CLAVIER DU TO 7

Le clavier, incorporé sur la face avant du boîtier, est plan, lisse, compact ; sa durée de vie en sera très certainement prolongée. De plus, ce choix permettra à des cousins de votre ordinateur TO7 d'être installés dans des locaux publics (écoles, clubs...) où ils seront mis à très rude épreuve.

Possédant 57 touches à répétition, le clavier de votre TO7 vous permettra de générer les lettres majuscules et minuscules, les chiffres et les caractères spéciaux (ponctuation, opérations...). Cette présence des minuscules est exceptionnelle pour le prix de votre système.

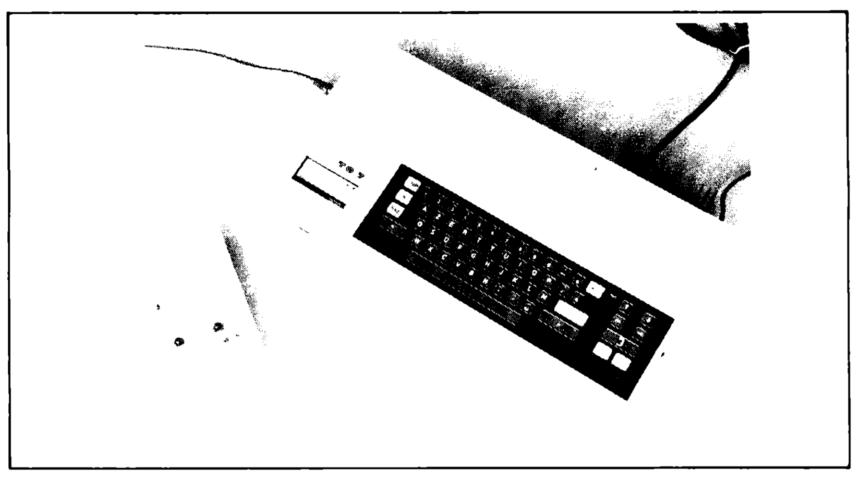


FIGURE 1 : CLAVIER DU TO7

Pour taper des lettres minuscules, appuyez sur la barre d'espace et sur l'une des touches Shift. Une petite lumière rouge s'allume alors en haut et à droite du clavier. Pour revenir ensuite en majuscules, tapez à nouveau sur les deux mêmes touches et la lumière rouge s'éteindra.

Le clavier de votre TO7 comporte en plus à droite du clavier quatre touches de déplacement du curseur vers la droite, la gauche, le haut et le bas. Ces touches sont indispensables pour corriger les erreurs de frappe et revenir en arrière pour retaper les bons caractères.

La disposition des caractères sur le clavier est celle des machines à écrire françaises (c'est-à-dire un agencement AZERTY). Ceci permettra à de nombreuses personnes de ne pas être dépaysées au premier contact avec le clavier du TO7.

1.3 SORTIE VIDÉO

Le TO7 se connecte sur les téléviseurs couleur par l'intermédiaire d'une prise spéciale Péritel, installée sur tous les téléviseurs couleur produits en France depuis 1980. Cette prise permet une excellente sortie vidéo avec netteté de l'image et des couleur.

Si vous ne disposez pas de cette prise Péritel sur votre téléviseur, il est généralement facile de la faire installer par un réparateur de télévision. Une autre solution, plus coûteuse et de moindre qualité, consiste à acheter un codeur Péritel/Sécam ou Péritel/Pal.

Pour connecter votre TO7 à une télévision par la prise Péritel, tournez votre système et regardez son arrière. Deux câbles en sortent ; ce sont la prise secteur (câble fin) et la prise Péritel (câble épais). Prenez ce dernier câble en main.

Tournez alors votre téléviseur et cherchez la prise femelle Péritel. Lorsque vous l'avez trouvée, prenez la prise au bout du câble et insérez-la.

Mettez alors en fonctionnement votre téléviseur et sélectionnez une des chaînes couleur (U.H.F.). C'est tout ; vous n'avez rien d'autre à régler.

Le micro-ordinateur Thomson TO7 utilise l'écran de la télévision soit pour afficher un texte ou pour faire du graphisme haute-résolution. Dans le premier cas, vous aurez 24 lignes de 40 caractères. En mode graphique, la définition obtenue pour l'écran est de 320 points horizontaux sur 200 points verticaux, c'est-à-dire un total de 64 000 points. Le rapport du nombre de points à son prix est excellent pour le TO7, et cela est un de ses principaux avantages.

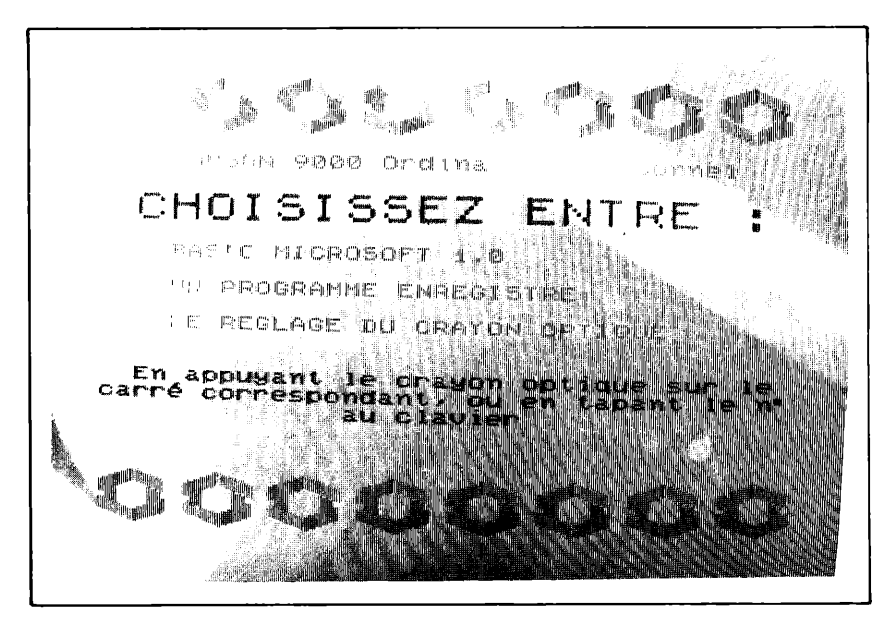


PHOTO B : GRAPHISME HAUTE RESOLUTION SUR TO7

1.4 CRAYON LUMINEUX

Au dessus du clavier du TO7, vous trouverez le crayon lumineux de votre ordinateur personnel dans un emplacement spécial. Cette possibilité est exceptionnelle pour les micro-ordinateurs dont le prix est du même ordre que celui du TO7. Avec votre crayon lumineux, vous pourrez sélectionner un choix dans un menu, dessiner sur l'écran, etc...



FIGURE 2 : UTILISATION DU CRAYON LUMINEUX

Pour utiliser le crayon lumineux, vous devrez le sortir de son étui, puis approcher l'écran de télévision avec le devant du crayon comme le montre la figure 2 . En effet, le stylo lumineux est composé d'une cellule sensible à la lumière qui permet de repérer ses coordonnées sur l'écran. Toutefois, la couleur rouge est équivalente au noir. C'est la raison pour laquelle les carrés sur lesquels vous devrez pointer le crayon lumineux sont de couleur bleu clair, en général.

Thomson accorde beaucoup d'importance au crayon lumineux. Il sera utilisé dans de nombreux logiciels disponibles en cassettes MEMO 7.

1.5 CASSETTES PRÉENREGISTRÉES MÉMO 7

Sur le côté gauche du clavier de votre TO7 est situé un lecteur de cartouches spéciales appelées MEMO 7. Tous les logiciels vendus pour le Thomson TO7 sont vendus sous forme de MEMO 7, car ils sont ainsi incopiables.

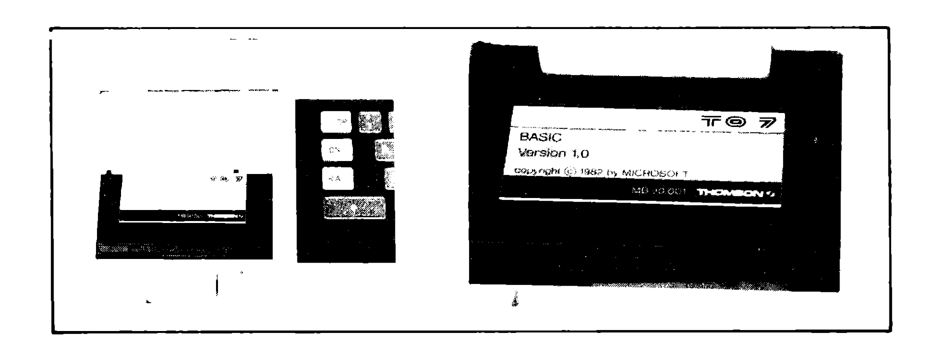


FIGURE 3 : UTILISATION D'UNE CARTOUCHE MEMO 7

Avant de mettre en route votre TO7, vérifiez toujours qu'une cartouche MEMO 7 est logée dans le compartiment prévu à cet effet. Dans le cas contraire, votre ordinateur personnel ne fonctionnerait pas. Pour introduire une cartouche MEMO 7, il suffit de soulever le couvercle, de mettre la cartouche dans le bon sens (connecteur métallique vers le TO7), puis de refermer l'ensemble.

1.6 STRUCTURE INTERNE

Pour utiliser votre ordinateur personnel TO7, vous n'avez besoin d'aucune connaissance préalable en électronique. Un ordinateur personnel est composé de circuits intégrés, ou "puces électroniques", nées il y a dix ans à peine. Si vous ouvrez votre micro-ordinateur TO7, vous verrez des boîtiers noirs en céramique munis de pattes métalliques ; ce sont les circuits intégrés.

Mais comment fonctionne un ordinateur ? Un ordinateur personnel, tel le TO7, est avant tout une machine à traiter l'information. Elle reçoit, stocke, traite et communique des données avec le clavier, le téléviseur couleur, ou avec un magnétophone à cassettes.

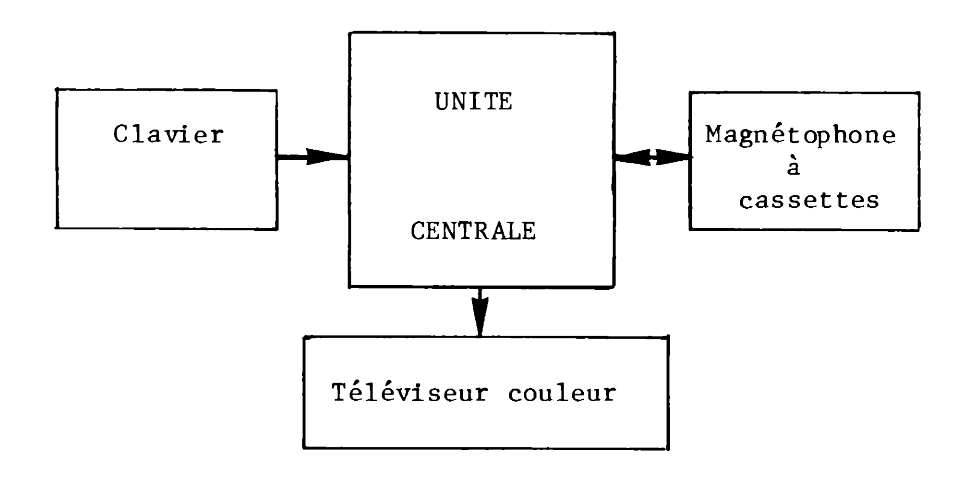


FIGURE 4 : STRUCTURE DU TO7

Le coeur de votre ordinateur personnel est appelé unité centrale (en anglais Central Processing Unit, ou CPU en abrégé). Cette unité centrale supervise le fonctionnement du système TO7 tout entier.

Elle décide quand aller chercher les données au clavier ou les envoyer sur l'écran, sur la cassette... Elle exécute les programmes que vous lui demandez.

L'unité centrale est organisée autour d'une puce - appelée microprocesseur - qui fait des calculs, va chercher les données et les range. Pour pouvoir travailler assez rapidement, le microprocesseur ne va pas demander une à une les données au clavier ; il a besoin pour cela d'autres circuits, appelés mémoires, qui contiennent le programme et la plupart des données utiles à celui-ci.

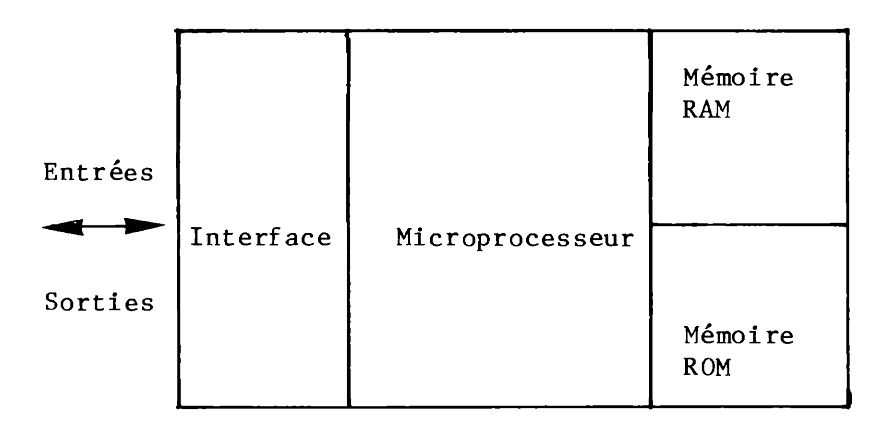


FIGURE 5 : UNITE CENTRALE DU TO7

Deux types de mémoires existent et sont représentées sur la figure 5 :

- mémoire ROM (Read Only Memory) ou mémoire morte ;
- mémoire RAM (Random Access Memory) ou mémoire vive.

Une mémoire morte est un ensemble d'instructions, de données... dont le contenu est permanent, n'est pas effacé lorsque l'on coupe le courant, et qui contient par exemple des programmes de communication avec le clavier. Le contenu des mémoires mortes du TO7 est défini par Thomson à la conception de la machine et vous ne pourrez y toucher. Les cassettes MEMO 7 décrites précédemment sont constituées de mémoires mortes.

Dans une mémoire RAM (ou vive), on peut écrire, lire ou effacer de l'information aussi souvent qu'on le souhaite ; elle contiendra, par exemple, les programmes que vous taperez, mais elle sera effacée par toute coupure de courant.

Le microprocesseur du TO7 est un Motorola 6809 de conception récente et d'une vitesse importante par rapport à ses concurrents 8 bits. La taille mémoire du TO7 est de 28 000 caractères en standard, mais peut être étendue de 16 000 caractères si besoin est. La mémoire vive (RAM) contient 22 000 caractères dont 8 000 sont disponibles pour vous et 14 000 sont réservés à la gestion de l'écran. La mémoire ROM contient, quant à elle, 6 000 caractères.

Outre le microprocesseur et les mémoires, votre ordinateur personnel TO7 contient l'électronique suivante :

- gestion du clavier
- gestion du crayon lumineux
- gestion de la télévision

ainsi que trois connecteurs d'extension du système situés sur la face arrière.

Nous devons encore introduire les notions d'octet et de bit. Un ordinateur devant gérer des informations, il doit les coder de manière interne pour pouvoir travailler. Etant constitué de composants électroniques, l'ordinateur code les données sur deux niveaux de tension qui sont 0 volt (0 logique) et 5 volts (1 logique). Lorsqu'il "examine" une tension, il teste une donnée élémentaire ou encore un bit (0 ou 1). Pour représenter des données plus complexes, on groupe les bits en octets (1 octet = 8 bits). Vous devez seulement retenir qu'un caractère est représenté par un octet.

1.7 AUTRES EXTENSIONS DISPONIBLES

Au dos de votre ordinateur TO7 sont disponibles plusieurs emplacements pour des extensions.

Nous vous présentons ci-dessous les principales extensions disponibles actuellement.

Le lecteur de cassettes MEMO 7 ne permettant pas à l'utilisateur classique de sauvegarder ses programmes, Thomson a développé spécialement pour le TO7 un lecteur/enregistreur de programmes, magnétophone bipiste (sons et données) que vous devrez acquérir pour programmer en BASIC par exemple.

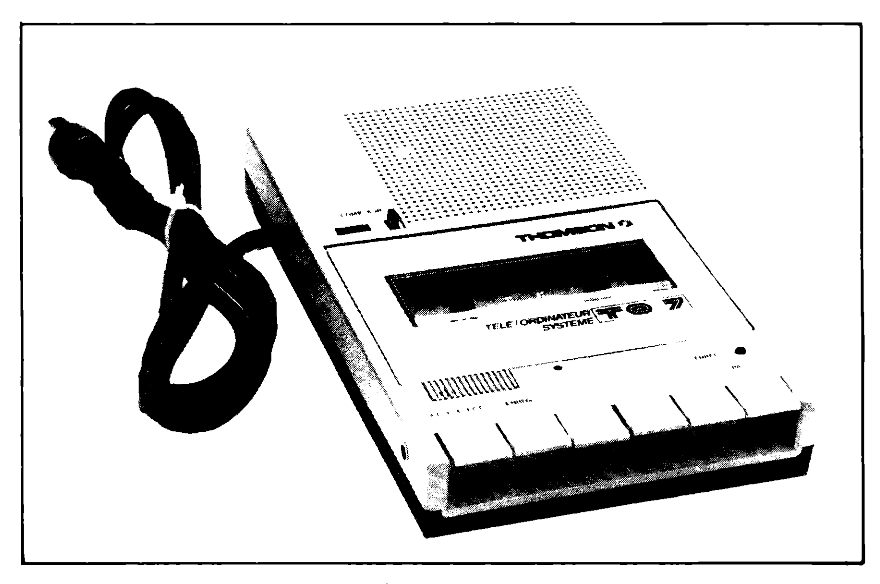


PHOTO C : LECTEUR/ENREGISTREUR DE PROGRAMME

Trois connecteurs d'extension sont situés à l'arrière du TO7. Un de ces connecteurs est réservé à l'extension de 16 000 octets de la mémoire vive (RAM) destinée à l'utilisateur.

Les autres connecteurs peuvent être utilisés pour les utilisations suivantes :

- Utilisation de disquettes

Celle-ci est réalisée par l'intermédiaire d'un contrôleur extra-plat relié par un câble à un des connecteurs du TO7, gérant jusqu'à quatre unités de disquettes de 80 000 octets formatés. Les lecteurs de disquettes s'enfichent les uns sur les autres, les connexions s'effectuant à cette occasion.

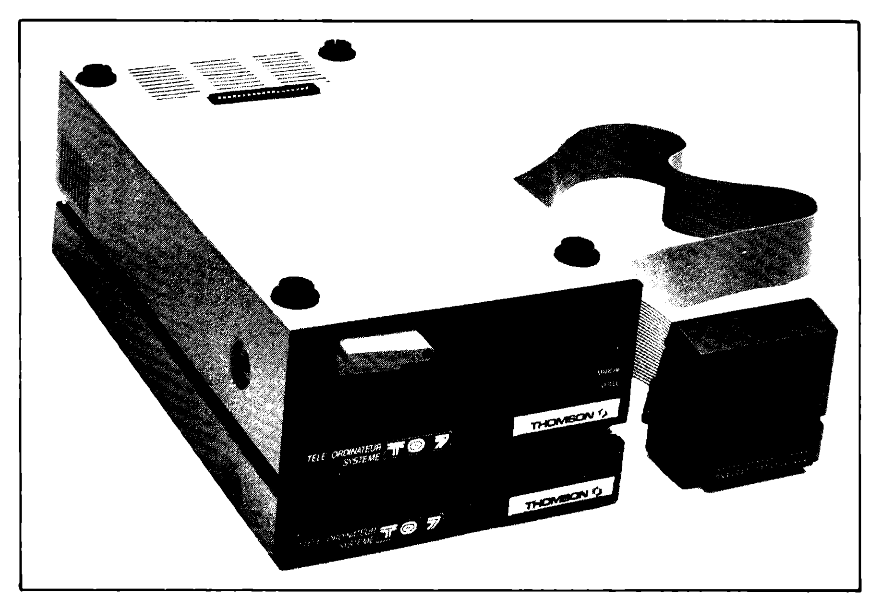


PHOTO D : CONTROLEUR ET UNITE DE DISQUETTES

- Communication avec des périphériques

L'interface de communication se connecte dans l'un des trois logements du TO7. Elle comporte les éléments suivants :

* Interface parallèle du type Centronics permettant de connecter de nombreuses imprimantes (Centronics, Epson, Thomson...).

* Interface série asynchrone RS232C dont la vitesse est programmable de 110 à 4 800 bits/s, et qui permet de communiquer avec un autre TO7, avec un réseau télématique par l'intermédiaire d'un modem ou avec un terminal.

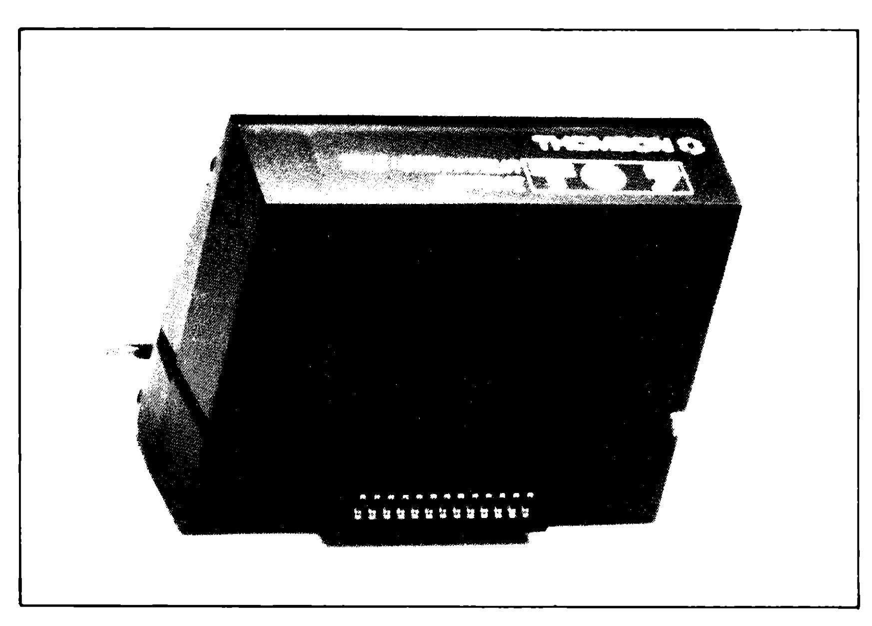


PHOTO E : CONTROLEUR DE COMMUNICATION

1.8 MISE EN ROUTE DU TO 7

Pour commencer à travailler, vous devez faire un certain nombre de branchements :

- branchement du TO7 au téléviseur
- branchement du TO7 au secteur 220 volts
- branchement du TO7 au magnétophone à cassettes si nécessaire

Ensuite, ne mettez pas encore en route ! Vérifiez qu'une cartouche MEMO 7 est placée dans le compartiment spécial prévu à cet effet.

Enfin, et seulement maintenant, mettez en route votre téléviseur et votre TO7. L'écran doit apparaître sans aucun parasite. Dans le cas contraire, vérifiez le branchement du téléviseur.

Si la cartouche MEMO 7 est celle du BASIC, vous verrez alors apparaître l'écran suivant :



Sortez alors votre crayon lumineux et choisissez l'option qui vous convient.

CHAPITRE **2**

Initiation au BASIC par le jeu

2.1 INTRODUCTION

Découvrir la micro-informatique, c'est pénétrer dans un monde fascinant. Un peu d'expérience et de pratique vous permettront d'en entreprendre l'exploration avec votre Thomson TO7.

Que vous soyez passionné de mathématiques, de gestion, de graphisme, de jeux ou même de musique, vous pourrez pratiquer sans contrainte votre discipline préférée. Il est impossible de dresser une liste exhaustive des innombrables applications de la micro-informatique ; les seules limites que vous rencontrerez seront celles de votre imagination...

Le but de ce chapitre est de vous montrer quelques-uns des principes fondamentaux de la programmation et leur mise en oeuvre pour la création d'un jeu en BASIC. Vous initier tout en vous divertissant, telle est donc notre ambition.

2.2 PROGRAMMATION ET LANGAGE BASIC

2.2.1 Généralités

Les progrès de l'informatique et le nombre croissant de ses applications ont contribué à créer au niveau du grand public le mythe de l'Ordinateur.

Il est bien sûr impressionnant de voir un ordinateur jouer merveilleusement bien aux échecs ou résoudre en une fraction de seconde des problèmes mathématiques très complexes. Admirer cet ordinateur, c'est toutefois commettre une erreur aussi grossière que d'admirer un violon et non le virtuose qui en joue.

L'ordinateur n'est qu'un instrument dans les mains de l'homme. De même qu'un violon prend vie dans les mains d'un musicien, un ordinateur ne doit son "intelligence" qu'à son programmeur. Livré à lui-même, un ordinateur n'est qu'un fatras de circuits intégrés. Seule l'action de l'homme peut permettre à cet assemblage de réaliser une fonction donnée, jouer aux échecs par exemple.

Il est fondamental d'insister sur ce fait, afin que vous compreniez bien que la mise en oeuvre des possibilités de votre micro-ordinateur TO7 dépend entièrement de vous, et de vous seul. C'est à vous de le programmer intelligemment pour lui faire exécuter exactement ce que vous désirez. Si vous voulez faire jouer votre TO7 aux échecs, il faut que vous écriviez le programme qui lui permettra de le faire. Lorsque vous verrez votre micro-ordinateur jouer, vous saurez alors que c'est votre programme qui est en train de "tourner", et vous qui jouez, en définitive, par son intermédiaire.

Cette totale dépendance de la machine envers son programmeur a bien sûr quelques inconvénients :

. Si vous omettez ne serait-ce qu'une virgule, un espace ou même un guillemet, votre TO7 sera totalement désorienté, incapable de comprendre ni d'exécuter ce que vous lui demandez.

. Vous ne pourrez jamais accuser votre TO7 de se tromper car il en est incapable. Les erreurs ne peuvent provenir que de vous, de même que les fausses notes sont dues au musicien et non pas à son instrument. Inutile donc d'être de mauvaise foi et injuste avec votre micro-ordinateur, il n'y est pour rien et de plus tout cela lui est égal...

Ces "inconvénients" sont mineurs en comparaison des avantages que procure la maîtrise d'un micro-ordinateur. Au gré de votre fantaisie, le Thomson TO7 se transformera en un agréable et infatigable compagnon de jeux, en un mathématicien modèle ou même en un dessinateur doué ! La programmation vous ouvre des horizons infinis.

Plus rien ne vous limitera quand vous saurez qu'il est impossible d'endommager un micro-ordinateur par une mauvaise programmation. Des messages d'erreurs vous signaleront vos fautes éventuelles. Le pire qui puisse vous arriver est d'être obligé de réinitialiser le système, ce qui n'est tout de même pas dramatique.

Alors n'hésitez pas à taper sur le clavier tout ce qui vous passe par la tête. Dès qu'une idée vous vient, essayez donc de la réaliser. De nombreux exemples sont donnés dans ce livre, utilisez-les et améliorez-les.

Nous espérons que bientôt vous éprouverez autant de plaisir que nous à programmer, et que vous saurez très vite exploiter au mieux les énormes possibilités de votre TO7.

2.2.2 Programmation et langage BASIC

* QU'EST-CE DONC QU'UN PROGRAMME ?

Le paragraphe précédent a mis l'accent sur l'importance de la programmation. Il reste maintenant à définir la notion de programme.

Un programme est une suite d'instructions que l'utilisateur souhaite voir exécuter dans un ordre déterminé pour réaliser une certaine fonction, jouer aux échecs par exemple.

* COMMENT PEUT-ON PROGRAMMER UN MICRO-ORDINATEUR ?

Un ordinateur ne raisonne qu'en termes de courant (ou de tension) ; le courant passe (niveau haut) : 1, ou ne passe pas (niveau bas) : 0. Il existe dans un langage propre à la machine (que l'on appelle "langage machine") qui permet de donner, par une suite de 1 et de 0, des ordres exécutables par l'ordinateur. Cette façon de procéder s'avère généralement longue et complexe.

Des langages évolués de programmation ont été développés afin de permettre une programmation simple et rapide. Des interpréteurs ou des compilateurs traduisent ensuite les programmes écrits en langage évolué, en langage machine exécutable par l'ordinateur.

Le Thomson TO7 utilise le langage de programmation BASIC (Beginners All-purpose Symbolic Instruction Code). Mis au point au cours des années 60, ce langage est devenu au fil des ans de plus en plus puissant et s'est répandu dans le monde entier. Votre TO7 dispose de l'une des plus puissantes versions actuelles du BASIC. Il s'agit du BASIC niveau 5 Microsoft, développé spécialement pour le TO7 (version 1.0).

* COMMENT APPREND-ON UN LANGAGE DE PROGRAMMATION ?

L'apprentissage du BASIC est très semblable à l'apprentissage d'une langue étrangère. Il est nécessaire d'acquérir un certain vocabulaire et de connaître les règles de syntaxe du langage.

Ne vous faites pas de souci, le BASIC est un langage très facile à apprendre et très simple à utiliser. Son vocabulaire n'est pas très étendu ; les règles de grammaire sont précises, strictes et ne comportent pas les multiples exceptions qui rendent si difficile l'étude d'une langue étrangère.

Le chapitre suivant de ce livre donne une analyse détaillée du BASIC disponible sur le Thomson TO7. Vous pouvez vous y reporter pour des détails précis, ce qui suit n'ayant pas la prétention d'être plus qu'une présentation générale du BASIC.

2.2.3 Démarrage du BASIC

Le BASIC est disponible sous forme d'une cartouche "MEMO 7" enfichable dans l'unité de base du TO7. Pour l'utiliser, procédez de la façon suivante :

- enfichez la cartouche "MEMO 7" dans le logement prévu à cet effet
- mettez sous tension le téléviseur et le TO7
- répondez 1 au menu affiché sur l'écran du téléviseur

Le message

"BASIC Version 1.0 (C) MICROSOFT 1982"

apparaît alors sur l'écran.

Le symbole "OK" apparaît aussi, indiquant que l'interpréteur BASIC est prêt à fonctionner. Votre TO7 est prêt à servir en mode calculateur ou en mode programme.

2.3 MODE CALCULATEUR - MODE PROGRAMME

2.3.1 Mode calculateur

Le mode calculateur, appelé aussi mode immédiat ou mode commande, ressemble beaucoup au mode de fonctionnement d'une calculatrice. Ce mode permet en effet de demander à votre TO7 d'exécuter immédiatement un calcul, une instruction ou une commande.

* Tapez, par exemple, PRINT "BONJOUR" puis pressez la touche [ENTREE] : le caractère " s'obtient en pressant simultanément [.] et [" 2].

Vous remarquez que le message BONJOUR apparaît aussitôt sur l'écran de votre téléviseur. Que s'est-il-passé ?

Le TO7 dispose d'un interpréteur BASIC qui lui permet de décoder et d'exécuter les ordres qui lui sont donnés en BASIC. Le symbole OK signifie que l'interpréteur BASIC est actif.

A chaque fois que vous avez tapé l'un des caractères de l'instruction PRINT "BONJOUR", le TO7 a rangé ce caractère en mémoire tout en le faisant apparaître sur l'écran du téléviseur.

Lorsque vous avez pressé la touche [ENTREE], l'interpréteur BASIC s'est mis au travail. Ressortant de la mémoire les uns après les autres les caractères entrés précédemmment, l'interpréteur a reconnu la commande PRINT qui lui ordonne d'afficher un message sur l'écran du téléviseur (PRINT signifie imprimer en anglais). Il a ensuite reconnu entre les guillemets le message à afficher.

Ayant ainsi totalement décodé l'ordre donné, l'interpréteur BASIC a pu passer à la phase d'exécution, ce qui explique l'apparition immédiate du message BONJOUR sur l'écran.

Le symbole OK signifie que l'interpréteur BASIC est à nouveau prêt pour recevoir et exécuter tout calcul ou instruction.

* Exécutons quelques calculs élémentaires en mode calculateur.

Par exemple :

PRINT 3 + 2 [ENTREE]

L'interpréteur BASIC reconnaît l'ordre d'afficher immédiatement sur l'écran du téléviseur le résultat de l'addition des deux nombres 3 et 2.

Ce résultat apparaît donc aussitôt : 5 suivi du classique OK.

donne sur l'écran

16.8571

Le TO7 permet ainsi de calculer la valeur d'expressions mathématiques très complexes. Pour des détails plus complets sur les opérateurs arithmétiques et leur hiérarchie, reportez-vous au chapitre suivant.

Remarque : l'opérateur ↑, élévation à la puissance (X ↑ Y signifie X à la puissance Y), est obtenu en pressant simultanément les touches [.] et [ȃ].

2.3.2 Mode programme

Le deuxième mode de fonctionnement du TO7 est le mode programme. Ce mode permet, comme son nom l'indique, de mettre en mémoire des programmes écrits en BASIC.

* ECRITURE D'UN PROGRAMME

Une ligne de programme est constituée par un numéro de ligne suivi d'une ou de plusieurs instructions.

Sur le TO7 le numéro de ligne devra obligatoirement être un nombre entier compris entre 0 et 63 999. La ligne ne devra pas comporter plus de 255 caractères.

- Tapez

```
10 PRINT "BONJOUR" [ENTREE]
```

Cette fois le message BONJOUR n'apparaît pas sur l'écran du téléviseur. Contrairement à ce qui s'était passé en mode calculateur, la pression de la touche [ENTREE] n'a pas mis l'interpréteur BASIC en action. L'introduction du numéro de ligne 10 a en effet indiqué au TO7 qu'il s'agissait d'une ligne de programme devant être stockée en mémoire et non pas d'une instruction exécutable immédiatement.

- Tapez

```
RUN [ENTREE]
```

Le message BONJOUR apparaît alors sur l'écran du téléviseur. La commande RUN demande à l'interpréteur BASIC d'exécuter le programme qui se trouve en mémoire. Ce programme est constitué pour l'instant par la seule ligne 10 PRINT "BONJOUR". Cette ligne est donc décodée et exécutée.

Le symbole OK réapparaît, indiquant ainsi la fin de l'exécution du programme et la disponibilité du TO7 pour recevoir de nouvelles commandes.

- Tapez

```
5 INPUT "QUEL EST VOTRE PRENOM  " ; P$
  [ENTREE]
15 GOTO 5 [ENTREE]
```

- Tapez

```
LIST [ENTREE]
```

Cette commande ordonne au TO7 d'afficher sur l'écran le programme contenu dans la mémoire.

Vous voyez alors :

```
 5 INPUT "QUEL EST VOTRE PRENOM  " ; P$
10 PRINT "BONJOUR"
15 GOTO 5
```

Bien que mises en mémoire après la ligne 10, les lignes 5 et 15 sont rangées par le TO7 dans l'ordre croissant des numéros de lignes.

- Faites maintenant

RUN

Vous avez certainement compris que la pression de la touche [ENTREE] est indispensable pour valider une ligne de programme ou une instruction. Nous ne vous indiquerons donc plus explicitement qu'il faut presser cette touche mais n'oubliez pas pour autant de le faire !

La commande RUN, ayant pour effet de lancer l'exécution du programme, la question "QUEL EST VOTRE PRENOM ?" s'affiche sur l'écran.

Remarque : le point d'interrogation est placé automatiquement après la question car le délimiteur utilisé est un point-virgule. Pour éviter ce point d'interrogation, inutile dans certains cas, il suffit de mettre une virgule en tant que délimiteur.

Répondez à la question posée (sans oublier de presser [ENTREE] pour valider votre réponse). Le TO7 vous dit alors poliment bonjour puis vous redemande aussitôt votre prénom.

* ARRET DE PROGRAMME

Le programme précédent ne peut s'arrêter tout seul en raison du saut inconditionnel GOTO de la ligne 15. C'est donc à l'utilisateur d'en arrêter l'exécution.

En pressant la touche [STOP], on peut "stopper" momentanément l'exécution d'un programme. La relance du programme se fera en appuyant sur une quelconque des touches du clavier à l'exception de [CNT], [.] et bien sûr [STOP].

On peut aussi presser simultanément les touches [CNT] et [C]. Dans ce cas le message BREAK ON n° de ligne apparaît. La relance du programme n'est alors possible que par la commande RUN ou la commande CONT (abréviation de CONTinuer).

Avant de passer à la suite, arrêtez donc l'exécution du programme en pressant simultanément [CNT] et [C].

* MODIFICATION DU PROGRAMME

Vous aimeriez peut-être que le TO7 vous dise bonjour en vous appelant par votre prénom. Il faut pour cela modifier la ligne 10 pour la transformer en :

```
10 PRINT "BONJOUR_" ; P$
```

(_ représente un espace).

Plusieurs méthodes sont possibles :

- Tapez simplement

```
10 PRINT "BONJOUR_" ; P$
```

Cela aura pour effet d'"écraser" l'ancienne ligne 10 par la nouvelle.

- Vous pouvez aussi utiliser l'éditeur "plein écran" du TO7.

Les quatre touches [→], [←], [↑], [↓], vous permettent de placer le curseur sur la ligne à modifier et de l'amener exactement là où des modifications doivent être faites.

Amenez ainsi le curseur sous le guillemet qui suit la lettre R. Pressez ensuite la touche [INS], ce qui a pour effet d'insérer un espace entre le R et le guillemet, comme cela était voulu.

Déplacez ensuite le curseur jusqu'à la fin de la ligne et tapez ; P$. N'oubliez pas de valider la ligne en pressant la touche [ENTREE].

- Faites maintenant LIST et vérifiez que le programme est maintenant conforme à vos désirs.

- Vous pouvez en lancer l'exécution avec RUN. Si un message d'erreur survient (? SN ERROR par exemple), corrigez la ligne incriminée à l'aide des méthodes exposées ci-dessus.

Vous disposez également comme moyen de correction de la touche [EFF] qui permet de supprimer le caractère situé au-dessus du curseur.

Remarque : si vous maintenez la pression sur une touche pendant plus d'une seconde environ, il y a répétition automatique du caractère donné par cette touche. Cela peut être mis à profit pour le déplacement du curseur.

* INSERTION DE LIGNES - SUPPRESSION DE LIGNES

- Une bonne habitude à prendre dès le début est de numéroter les lignes d'un programme que l'on est en train d'écrire de dix en dix par exemple. Cela permet d'insérer par la suite de nouvelles lignes dans le programme.

Le programme précédent est :

```
 5 INPUT "QUEL EST VOTRE PRENOM" ; P$
10 PRINT "BONJOUR_" ; P$
15 GOTO 5
```

On peut donc insérer :

```
 7 PRINT
12 PRINT
```

Faites LIST et vous aurez alors :

```
 5 INPUT "QUEL EST VOTRE PRENOM" ; P$
 7 PRINT
10 PRINT "BONJOUR_" ; P$
12 PRINT
15 GOTO 5
```

Les deux lignes rajoutées ont pour intérêt d'aérer la présentation sur l'écran du téléviseur.

- Si vous désirez voir ce programme ne s'exécuter qu'une fois, la ligne 15 devient totalement inutile. Vous pouvez la supprimer en tapant simplement 15 puis [ENTREE]. La suppression d'une ligne se fait simplement en tapant son numéro puis [ENTREE].

2.4 PRINCIPALES INSTRUCTIONS BASIC

2.4.1 Constantes et variables

* CONSTANTES

Le BASIC permet d'utiliser des constantes numériques et des constantes "chaînes".

Les constantes numériques peuvent être de type entier, réel ou double précision mais ce n'est pas fondamental pour une première approche. Des détails plus précis (notations, plage de variations, etc...) seront donnés dans le chapitre suivant.

Exemple : 18 est une constante réelle.

Une constante "chaîne" est une suite de caractères encadrée par des guillemets.

Exemple : "BONJOUR" est une constante "chaîne".

* VARIABLES

Une variable est un emplacement particulier de la mémoire auquel le programmeur peut donner le nom qu'il désire. Cet emplacement peut voir son contenu "varier", d'où le nom de variable.

Les noms de variables peuvent comporter de 1 à 40 caractères, le premier caractère étant obligatoirement une lettre, les autres peuvent être des lettres ou des chiffres. Cela permet de donner aux variables des noms "parlants" qui permettent de savoir ce que représente exactement la variable considérée.

Dans un jeu, par exemple, il est parfois nécessaire de créer une variable SCORE qui retiendra le résultat du joueur, une variable RECORD qui contiendra le meilleur score réalisé, etc...

SCORE et RECORD sont donc des noms possibles de variables.

2TAB est impossible car ce nom commence par un chiffre alors que TAB2 est possible.

Un nom de variable ne doit être ni un mot-clé du langage ni commencer par un mot réservé du langage. Ainsi le nom TOTO est impossible car commençant par TO (instruction FOR - TO - NEXT).

Seuls les seize premiers caractères du nom d'une variable sont réellement pris en compte. Deux variables dont les noms commencent par seize caractères identiques seront considérés par le BASIC comme identiques.

```
MISE     )
         ) sont des noms de variables différentes.
MISERE   )
```

```
ABCDEFGHIJKLMNOPX    )
                     ) représentent la même variable.
ABCDEFGHIJKLMNOPZ    )
```

Les variables peuvent avoir, comme les constantes, différents types : entier, réel, double précision et chaîne. Retenons surtout pour l'instant les variables "chaînes" dont le type est indiqué par le caractère $, placé à la fin du nom de la variable. Ainsi A$, REPONSE$, sont des noms de variables "chaînes".

* TABLEAUX

Un tableau est un ensemble de variables de même type, référencé par un nom unique.

Chaque élément du tableau est représenté par le nom du tableau suivi de un ou plusieurs indices (il faut autant d'indices qu'il y a de dimensions au tableau).

Exemples : A(2) est un élément d'un tableau à une seule dimension.

T(3,5) est un élément d'un tableau T à deux dimensions.

Vous pourrez utiliser des tableaux ayant de 1 à 255 dimensions, chacun des indices pouvant prendre des valeurs de 0 à 32 767 (dans la mesure où la mémoire disponible le permettra).

Pour des tableaux ayant au plus deux dimensions avec des indices ne dépassant pas 10, il n'est pas nécessaire de déclarer ces tableaux. Cela est obligatoire pour des tableaux plus importants. La déclaration d'un tableau se fait à l'aide de l'instruction DIM.

Exemple : 10 DIM A(20) déclare le tableau A à une dimension et pouvant contenir 21 éléments A(0), A(1),..., A(20).

Remarque : il est fortement conseillé de déclarer les tableaux que l'on utilise même lorsque cela n'est pas obligatoire. Il est en effet dangereux de s'exposer à des fautes par simple paresse !

2.4.2 Instruction d'affectation LET, =

L'instruction d'affectation LET,= permet d'attribuer à une variable une valeur donnée.

Exemples :

LET A = 10 donne à la variable A la valeur 10
LET P$ = "BONJOUR" donne à la variable "chaîne" la valeur "BONJOUR".

Le LET est en fait optionnel. Les informaticiens, étant en général de grands paresseux (est-ce un défaut...?), ont en effet estimé qu'il n'était pas nécessaire d'indiquer LET pour exécuter une affectation. Cela a malheureusement pour effet de conduire à des expression du type I = I + 1. D'un point de vue purement mathématique cela est bien sûr très choquant car I = I + 1 entraîne 1 = 0, ce qui est une absurdité.

En fait il faut se souvenir que le LET est sous-entendu et que le = signifie, dans ce cas, affectation. I = I + 1 signifie simplement que la variable I est incrémentée d'une unité (nouvelle valeur de I = ancienne valeur de I + 1, ce qui équivaut aussi à LET I = I + 1).

Faites donc comme tout le monde et ne mettez pas le LET mais soyez vigilant ! Ne vous laissez pas aller à des simplifications conduisant à des erreurs monstrueuses.

Exemple : A = 2 * A + 1 est totalement différent de A = -1.

```
10 A = 0
20 A = 2 * A + 1
30 PRINT "A1_" ; A
40 A = -1
50 PRINT "A2_" ; A
60 END
```

(_ est un espace)

L'exécution de ce petit programme fait apparaître sur l'écran du téléviseur

```
A1   1
A2  -1

OK
```

Il eût été judicieux de créer un symbole spécial pour l'instruction d'affectation plutôt que d'utiliser le signe égal. Certains langages l'ont fait et permettent d'écrire, par exemple, I ← I + 1.

Remarque : si vous tapez, comme cela est vivement conseillé, les petits programmes d'exemples, n'oubliez pas de faire au préalable NEW pour effacer les anciens programmes de la mémoire, et [RAZ] pour effacer l'écran. Vous éviterez ainsi quelques réactions pour le moins bizarres de votre TO7.

2.4.3 Instructions d'entrée-sortie : INPUT, PRINT

Un micro-ordinateur est, comme tout ordinateur, une machine destinée au traitement de l'information. Le travail d'un micro-ordinateur comporte trois phases essentielles : l'acquisition des données, le traitement de ces données et l'affichage des résultats obtenus.

Les phases d'acquisition des données et de sortie des résultats sont les phases de "dialogue" entre l'utilisateur d'un programme et le programme lui-même. Ces phases nécessitent des instructions spéciales, appelées instructions d'entrée-sortie. En BASIC ces instructions sont INPUT et PRINT.

* INSTRUCTION D'ENTREE INPUT

L'instruction INPUT est fondamentale pour la phase d'acquisition des données. C'est elle en effet qui permet à l'utilisateur d'introduire des données à partir du clavier en cours d'exécution d'un programme.

Si par exemple vous désirez voir votre TO7 vous dire bonjour en vous appelant par votre prénom, il est nécessaire de fournir au TO7 votre prénom. Cela justifie la ligne 5 du premier programme de ce chapitre :

```
5 INPUT "QUEL EST VOTRE PRENOM _ "; P$
```

Lors de l'exécution de cette ligne 5, le TO7 affiche sur l'écran du téléviseur la question, constituée par le message placé entre guillemets, et attend votre réponse. La réponse que vous lui donnez est alors rangée dans la variable "chaîne" P$.

Les applications de l'instruction INPUT sont innombrables, ne serait-ce qu'en raison de la rareté des programmes ne comportant pas de phase d'acquisition de données. Les jeux sur ordinateur font énormément appel à cette instruction. Donnons quelques exemples qui sont utilisés ultérieurement.

Exemples :

Combien d'argent le joueur veut-il miser ?

```
10 INPUT "MISE" ; M
```

Quel niveau de jeu le joueur désire-t-il ?

```
10 INPUT "NIVEAU" ; N
```

* INSTRUCTION DE SORTIE PRINT

L'instruction PRINT est indispensable pour la phase de sortie des résultats obtenus à la fin du traitement demandé. Elle permet en effet d'obtenir, sur l'écran du téléviseur, les résultats de calculs, l'affichage de messages et de dessins variés, etc.

La ligne 10 PRINT "BONJOUR_"; P$ (_ est un espace) complète la ligne 5 donnée précédemment. Voilà donc rapidement fait un petit programme qui dit poliment bonjour.

Les jeux utilisent beaucoup l'instruction PRINT pour afficher toutes sortes de choses.

Exemples :

```
 120 PRINT "SCORE"; SCORE
1000 PRINT "VOUS AVEZ GAGNE !!!"
```

2.4.4 Instructions de branchement : GOTO, GOSUB

Un ordinateur exécute toujours un programme de façon séquentielle, c'est-à-dire par ordre de numéros de lignes croissants, sauf exceptions : sauts inconditionnels, tests, sous-programmes.

Les tests seront vus dans le paragraphe suivant. Intéressons-nous tout d'abord aux sauts inconditionnels et aux sous-programmes.

* INSTRUCTION DE SAUT INCONDITIONNEL : GOTO

L'instruction GOTO n permet de sauter directement de la ligne où est située cette instruction à la ligne de numéro n.

Cette instruction est souvent très utile bien qu'elle soit, à juste titre il faut bien l'avouer, remise en cause par les partisans de la programmation structurée. Utiliser abondamment l'instruction GOTO est céder à une certaine facilité qui peut se révéler nuisible. Un programme comportant un trop grand nombre de GOTO devient en effet incompréhensible et inextricable. Sa mise au point s'avère alors difficile et d'éventuelles modifications quasiment impossibles.

Le premier programme de ce chapitre donne une application simple de l'instruction GOTO et montre comment permettre à un programmme de tourner sans fin.

```
15 GOTO 5
```

c'est-à-dire n° ligne de fin GOTO n° ligne de début.

L'instruction GOTO est souvent utilisée en même temps que les instructions de test qui seront décrites plus loin. Il est en effet nécessaire, suivant le résultat d'un test, de passer à une ligne ou à une autre.

Donnons un exemple d'application de GOTO dans un programme de jeu :

```
  :
  :
100  PRINT "VOUS GAGNEZ !!!"
110  GOTO 1000
  :
  :
200 PRINT "VOUS PERDEZ !!!"
210 GOTO 1000
  :
  :
1000 INPUT "VOULEZ VOUS REJOUER" ; R$
```

*   INSTRUCTION D'APPEL D'UN SOUS-PROGRAMME : GOSUB

L'analyse d'un problème permet souvent de le diviser en plusieurs petits problèmes pouvant être résolus séparément, la mise en commun des solutions de chacun d'eux permettant ensuite de résoudre le problème initial. On peut donc écrire un programme principal et plusieurs "petits" programmes appelés sous-programmes (en fait souvent plus longs que le programme principal). Les sous-programmes sont appelés les uns après les autres par le programme principal afin de résoudre les différents petits problèmes, le programme principal se chargeant de trouver la solution globale.

Cette façon de procéder est appelée programmation structurée. Elle permet d'écrire des programmes modulaires, simples, clairs, justement mis au point module par module et aisément modifiables.

L'instruction d'appel d'un sous-programme est l'instruction GOSUB n où n est le numéro de la première ligne du sous-programme.

Le sous-programme doit se terminer par l'instruction RETURN afin de redonner le contrôle au programme principal dès qu'il a fini son travail.

Remarques :

- il est souvent utile de donner des noms aux différents sous-programmes que l'on peut être appelé à utiliser. Ce nom sera mis dans une instruction REM (REMarque) en tant que première ligne du sous-programme. Cela facilitera ensuite grandement la lecture du programme et permettra de savoir très vite quelles sont les fonctions réalisées par les différents sous-programmes.

- De façon génrale, n'hésitez pas à employer fréquemment l'instruction REM. Un programme bien commenté est nettement plus facile à comprendre qu'une suite d'instructions sans aucune explication.

Exemple : programme de calcul de la circonférence et de la surface d'un cercle connaissant son rayon.

```
 10  PI = 3,14
 20  INPUT "RAYON" ; R
 30  GOSUB 100
 40  GOSUB 200
 50  PRINT "CIRCONFERENCE = " ; C
 60  PRINT "SURFACE = " ; S
 70  END
100  REM CIRCONFERENCE
110  C = 2 * PI * R
120  RETURN
200  REM SURFACE
210  S = PI * R ↑ 2
220  RETURN
```

2.4.5 Instructions de test : IF. THEN, ELSE

Nous avons vu que le travail d'un ordinateur comporte trois phases distinctes : l'acquisition de données, le traitement des données et la sortie des résultats. Les instructions d'entrée-sortie ont été décrites, il reste maintenant à parler des instructions fondamentales du BASIC pour la phase de traitement des données.

Dans tout traitement il y a des calculs intermédiaires dont les résultats influent sur la suite du déroulement du programme. Il y a aussi des interventions de l'utilisateur (par des instructions INPUT) qui forcent le choix de telle ou telle option, jouer au niveau le plus difficile ou au niveau le plus facile, par exemple. Un programme doit donc avoir les moyens d'effectuer des tests et d'agir en fonction des résultats de ces tests.

L'instruction de test en BASIC est l'instruction IF /condition/ THEN /instruction 1/ ELSE /instruction 2/.

Cela peut se traduire en français par : SI la condition est réalisée (résultat non nul, expression logique "vraie") ALORS exécuter l'instruction 1 SINON exécuter l'instruction 2.

Le petit programme suivant met en évidence la plupart des modes d'utilisation de l'instruction de test.

```
 10  INPUT "A =" ; A
 20  INPUT "B =" ; B
 30  PRINT
 40  IF A = B THEN PRINT "A EGAL B" ELSE
     PRINT "A DIFFERENT DE B"
 50  PRINT
 60  IF A THEN PRINT "A EST NUL" : PRINT
 70  IF A * B < 0 THEN IF A < B THEN PRINT "A
     EST NEGATIF, B EST POSITIF" ELSE PRINT
     "A EST POSITIF, B EST NEGATIF"
 80  PRINT
 90  IF A > B OR A < 0 GOTO 10 ELSE GOTO 200
100  IF A < B AND B > 0 THEN GOTO 10
200  GOTO 10
```

- les lignes 10 et 20 permettent d'entrer les deux nombres A et B sur lesquels vont porter les tests.

- les lignes 30, 50 et 80 ont pour but d'aérer la présentation des résultats sur l'écran du téléviseur.

- la ligne 40 est une application typique de l'instruction de test et n'a pas besoin d'explication supplémentaire.

- la ligne 60 teste si A est nul. Si c'est le cas le message A EST NUL apparaît, sinon il y a passage à la ligne 70. On peut donc ainsi tester si le résultat d'un calcul est nul ou non et suivant la réponse du test exécuter une instruction ou une autre.

- la ligne 70 présente une possibilité extrêmement intéressante de l'instruction de test. On peut en effet mettre plusieurs tests en cascade.

- Si le produit A * B est positif il y a passage à la ligne 80 sans exécuter le reste de la ligne 70.

Par contre, si le produit A * B est négatif, alors le programme teste si A est plus petit que B. Si c'est le cas, le programme déduit que A est négatif et B positif (A * B < 0 entraîne A et B de signes différents, A < B entraîne A négatif et B positif), sinon c'est A qui est positif et B négatif.

La mise en route en cascade de tests est particulièrement intéressante pour optimiser la vitesse de certaines phases du traitement. Certains tests ne sont à effectuer que si le résultat d'un test précédent le permet. Il est donc inutile d'effectuer ces tests systématiquement. La mise en cascade est une solution élégante et rapide à ce problème.

- La ligne 90 introduit deux nouvelles notions : l'instruction IF sans l'instruction THEN et les expressions logiques.

L'instruction IF... GOTO... est identique à IF... THEN GOTO..., il n'y a pas lieu d'en dire plus.

Le test porte cette fois-ci sur une expression logique : A > B OR A < 0. Si A est plus grand que B ou si A est négatif, l'expression logique est dite "vraie" et l'instruction suivante est exécutée. Si A est plus petit que B et A est positif (expression logique contraposée de l'expression logique précédente) alors il y a passage directement à la ligne suivante.

- La ligne 100 donne un exemple d'utilisation de l'opération logique AND (c'est-à-dire ET). Si A est plus petit que B et si B est positif alors aller à la ligne 10.

2.4.6 Boucles FOR, TO, NEXT, STEP

Certains calculs en traitement doivent être exécutés plusieurs fois. Il est alors intéressant de créer une boucle qui sera parcourue autant de fois que cela est nécessaire. Les instructions FOR, TO, NEXT, STEP permettent de créer de telles boucles.

La syntaxe est la suivante :

FOR / variable numérique / = X TO Y STEP Z

200 NEXT / variable numérique /

La variable numérique qui suit le mot-clé FOR est la variable de contrôle de la boucle. Cette variable joue le rôle de compteur.

X est la valeur initiale du compteur, Y est la valeur finale.

L'expression STEP Z et la / variable numérique / qui suit le mot-clé NEXT sont facultatifs. Par défaut le pas (STEP) Z est pris égal à 1.

Comment fonctionne une boucle FOR - NEXT ?

- la variable de contrôle prend la valeur C
- la boucle est exécutée
- lorsque le programme arrive sur NEXT qui veut dire "passer à la valeur suivante de la variable de contrôle", il y a incrémentation de cette variable : variable de contrôle = variable de contrôle + Z (Z égale 1 par défaut).

Le programme teste ensuite si la variable de contrôle est plus petite que la valeur finale Y. Si c'est le cas il y a retour au début de la boucle, sinon le programme "sort" de la boucle et passe aux instructions suivantes.

Remarque : il est vivement recommandé, dans le cas de boucles imbriquées, de donner des noms différents à chacune des variables de contrôle des boucles imbriquées. Il faut aussi véfifier qu'à chaque instruction FOR correspond une instruction NEXT. N'oubliez pas non plus de préciser explicitement après un NEXT le nom de la variable de contrôle concernée bien qu'il soit en théorie possible de ne pas le faire. Ces diverses précautions pemettent d'éviter des erreurs de programmation parfois longues et difficiles à retrouver.

2.4.7 Premier jeu : REVERSE

Vous connaisez maintenant les instructions les plus importantes du BASIC. Vous êtes désormais potentiellement capable de programmer tout ce que vous voulez.

Le paragraphe suivant vous montrera comment créer un jeu en BASIC, de l'analyse du problème à l'écriture du programme en passant par le dessin d'un organigramme. Le programme ci-dessous a simplement pour but de vous montrer de quelle façon les instructions décrites précédemment peuvent être mises en oeuvre pour programmer un jeu. Nous espérons que vous tirerez profit de la lecture de ce programme, mais aussi que vous vous amuserez en l'utilisant.

* PROGRAMME DE JEU REVERSE

```
   5  REM PROGRAMME REVERSE
  10  DIM A(21) : CLS
  20  INPUT "ENTREZ UN NOMBRE N" ; N
  30  N = INT (ABS(N))
  40  FOR K = 1 TO N
  50  V = RND
  60  NEXT K
  70  FOR I = 1 TO 9
  80  R = INT (RND * 9 + 1)
  90  IF A(R)<>0 THEN GOTO 80
 100  A(R) = I
 110  NEXT I
 120  GOSUB 1000
 130  INPUT "ROTATION" ; ROT
 140  IF ROT > 9 THEN GOTO 130
 150  IF ROT = 0 THEN CLS : STOP
 160  FOR I = 1 TO INT (ROT/2)
 170  Z = A(I)
 180  A(I) = A(ROT - I + 1)
 190  A(ROT - I +1) = Z
 200  NEXT I
 210  GOSUB 1000
 220  FOR I = 1 TO 9
 230  IF A(I)<>I THEN GOTO 130
 240  NEXT I
 250  PRINT "GAGNE"
 260  END
1000  PRINT
1010  FOR I = 1 TO 9
1020  PRINT A(I) ;
1030  NEXT I
1040  PRINT "_ _ _ _" (4 espaces)
1050  PRINT
1060  RETURN
```

* REGLES DU JEU

Les chiffres 1, 2, 3, 4, 5, 6, 7, 8, 9 sont placés dans un ordre quelconque par le TO7.

Le but du jeu est d'ordonner ces chiffres par ordre croissant à partir de la gauche.

Pour ce faire vous devez demander au TO7 d'effectuer les inversions de votre choix.

Exemples :

* 1 3 5 7 9 2 8 4 6

ROTATION ? 4

7 5 3 1 9 2 8 4 6

* 2 3 4 5 1 6 7 8 9

ROTATION ? 4

5 4 3 2 1 6 7 8 9

ROTATION ? 5

1 2 3 4 5 6 7 8 9

GAGNE

Bien sûr le jeu refusera toute rotation supérieure à 9.

Au début du jeu un nombre N vous est demandé. Cela permet d'obtenir les séquences de nombres aléatoires différentes lors du tirage de la suite initiale. Deux nombres différents donnent deux suites initiales différentes.

Si vous désirez rejouer plusieurs fois avec une suite initiale donnée, il vous suffira de redonner à chaque fois le même nombre N au début du jeu pour obtenir cette suite initiale.

* REMARQUES

. Le but de ce jeu étant essentiellement de vous donner l'occasion d'étudier un programme, la structure du programme REVERSE n'est donnée qu'en annexe.

. De plus ce jeu est volontairement dépouillé de tout artifice de présentation. De nombreuses améliorations peuvent être faites : comptage du nombre de coups nécessaires pour ordonner la suite, couleurs, sons, affichage des règles du jeu, etc.... C'est un bon exercice que d'essayer de faire vous-même quelques-unes des modifications suggérées ci-dessus.

2.5 CRÉATION D'UN JEU : JEU DU 21

Nous avons vu que programmer un micro-ordinateur consiste à mettre dans la mémoire de ce micro-ordinateur une série d'instructions écrites dans un langage compréhensible par la machine. Vous connaissez maintenant les instructions fondamentales du langage évolué de programmation BASIC. Il reste encore à savoir comment passer d'un problème donné à l'écriture d'un programme qui résoudra ce problème.

Quand on leur soumet un problème, les gens cèdent très souvent à la tentation de jeter immédiatement sur une feuille une longue série d'instructions sans même savoir exactement ce qu'ils se proposent de faire. Une nouvelle idée vient à leur passer par la tête et voilà sur la feuille quelques lignes de plus sans préoccupation aucune de savoir si ces lignes sont compatibles ou non avec les lignes qui les précèdent.

Après de nombreuses heures perdues à tenter de mettre au point un programme qui, étant conçu en dépit du bon sens, refuse généralement de "tourner", ces gens-là sont obligés de tout effacer et de repartir à zéro.

Programmer doit être un plaisir, mais programmer sans méthode n'est qu'une perte de temps et une source d'erreurs.

La programmation comporte quatre étapes fondamentales qu'il convient d'effectuer systématiquement si on veut être efficace et rapide.

* analyse du problème,
* écriture d'un organigramme,
* codage,
* mise au point du programme.

Ces différentes étapes ont bien sûr plus ou moins d'importance suivant les problèmes posés. Afficher le message "BONJOUR" sur l'écran du téléviseur ne demande pas une analyse "poussée" ni même l'écriture d'un organigramme. Des problèmes plus complexes, comme c'est le cas pour la plupart des jeux sur micro-ordinateur, nécessitent par contre une analyse très fine et l'écriture d'un organigramme complet. C'est à ce prix qu'on peut être sûr d'aboutir à un programme qui "tourne" bien et remplit exactement le cahier des charges.

Nous allons montrer sur un exemple comment mettre en oeuvre les différentes étapes de la programmation en programmant un jeu très connu : le jeu du 21.

2.5.1 Analyse du jeu

L'analyse est la première étape de la programmation et on pourrait presque dire que c'est la plus importante. C'est d'elle que dépend en effet toute la suite des opérations.

L'analyse consiste dans un premier temps à définir exactement le problème que l'on veut résoudre. Quelles sont les question posées ? Quels résultats va-t-on en attendre ? Quelles sont les règles du jeu ? Quels sont les rôles respectifs du joueur et du micro-ordinateur ?

Le problème étant clairement posé, il s'agit ensuite de le décomposer en plusieurs sous-problèmes. Cette décomposition descendante se poursuit jusqu'à l'obtention de problèmes élémentaires ou "modules" ne nécessitant que des traitements simples.

Procédons à l'analyse du jeu du 21 que nous nous proposons de programmer.

* DEFINITION DU PROBLEME : REGLES DU JEU

Le jeu du 21 est un jeu de dés (ou de cartes) connu. Les règles que nous allons donner sont peut-être différentes de celles qui ont cours dans les casinos mais cela n'a que peu d'importance. L'essentiel est d'arriver à définir des règles précises et simples et d'obtenir un programme de jeu plaisant à utiliser.

. Le jouer lance un dé à six faces autant de fois qu'il le désire. La somme de ces lancers ne doit pas dépasser 21 mais on doit s'en approcher le plus possible.

. Si le total du joueur dépasse 21, le joueur perd.

. Lorsque le joueur estime un total suffisant, il peut arrêter de jouer le dé et c'est au TO7, qui joue le rôle de la banque, de lancer le dé à son tour.

. Si le total de la machine dépasse le total du joueur sans pour autant dépasser 21, le joueur perd.

. Si le total de la machine dépasse 21, le joueur gagne.

. Au début du jeu, le joueur se voit créditer d'un avoir, mille francs par exemple. A chaque tour le joueur est obligé pour pouvoir jouer de miser une certaine somme. S'il ne mise rien le jeu s'arrête.

. La mise ne doit pas dépasser l'avoir du joueur.

. Quand le joueur gagne, il peut gagner jusqu'à cinq fois sa mise.

. Lorsque le joueur perd, il perd sa mise.

. Si par malheur l'avoir du joueur s'annulait, le jeu s'arrête (un joueur ruiné n'a en effet plus les moyens de miser...). Cela n'est pas dramatique car toute relance du jeu recrédite l'avoir du joueur.

* RECAPITULATION DES CAS DE GAIN ET DE PERTE DU JOUEUR

. Le joueur gagne si la machine dépasse 21 en essayant de dépasser le total du joueur. Dans ce cas le joueur peut gagner jusqu'à cinq fois sa mise.

. Le joueur perd si son total dépasse 21 ou si la machine obtient un total supérieur à celui du joueur, sans dépasser 21. Dans ce cas le joueur perd sa mise.

. Si le joueur et le TO7 ont tous les deux 21, le coup est nul.

. Le jeu s'arrête si des revers successifs ont acculé le joueur à la ruine et réduit son avoir à zéro.

* DEROULEMENT DU JEU

. Il est nécessaire dans un premier temps de créditer l'avoir du joueur. C'est la phase d'initialisation.

. Le joueur peut lancer ensuite le dé autant de fois qu'il le désire. C'est la phase de jeu du joueur. Cette phase ne cesse que si le joueur le désire ou si, en voulant trop tenter sa chance, le joueur obtient un total dépassant 21.

. La machine lance le dé à son tour. C'est la phase de jeu de la machine.

. Enfin les cas de gain et de perte du joueur sont traités. L'avoir du joueur est ajouté à sa nouvelle valeur. Il y a ensuite soit retour au début pour un nouveau jeu, soit arrêt du programme pour cause de ruine du joueur.

Nous venons donc de définir quatre phases principales dont l'enchaînement permettra de jouer au jeu du 21 avec un micro-ordinateur TO7 dans le rôle de la banque.

2.5.2 Ecriture d'un organigramme

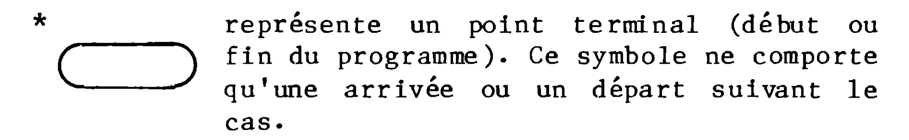
Ecrire ou plutôt dessiner un organigramme permet de visualiser les résultats de l'analyse.

Celle-ci a mis en balance plusieurs "modules". L'organigramme montre l'organisation de ces modules, les liaisons qui existent entre eux, les sauts, les branchements, les tests qui permettent de passer des uns aux autres afin de réaliser la fonction souhaitée.

Plusieurs organigrammes peuvent être dessinés. Un organigramme, dit organigramme principal, représente la suite logique des étapes nécessaires pour la résolution du problème. Plusieurs organigrammes secondaires représentent l'enchaînement des traitements nécessaires pour effectuer chacune des étapes intervenant dans l'organigramme principal.

Les organigrammes suivent donc une démarche descendante similaire à celle adoptée pour l'analyse. La décomposition s'arrête lorsque les traitements de niveau le plus bas peuvent être réalisés directement par les instructions du langage de programmation utilisé. Il sera alors temps de passer à la phase de codage du programme.

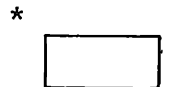
Dessiner n'importe comment un organigramme ferait perdre à celui-ci tout son intérêt. Des symboles spéciaux ont été mis au point pour dessiner les organigrammes. Quatre d'entre eux sont fondamentaux et interviennent très fréquemment.

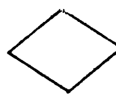


* représente un point terminal (début ou fin du programme). Ce symbole ne comporte qu'une arrivée ou un départ suivant le cas.

* représente une opération d'entrée-sortie (nécessité d'entrer une donnée avec INPUT ou d'afficher un message avec PRINT).

Ce symbole n'a qu'une arrivée et un départ.

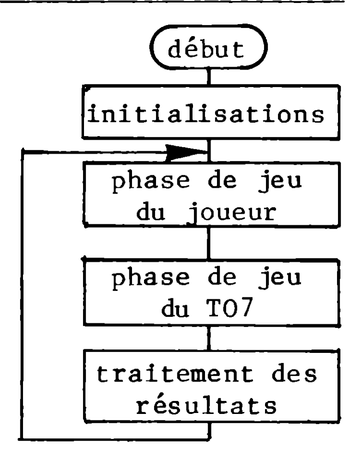
*  représente un ensemble de traitements à effectuer. Ce symbole ne comporte qu'une arrivée et un départ.

  représente un test (possibilité de prendre une décison en fonction du résultat du test). Ce symbole a une entrée et deux sorties l'une ou l'autre des sorties étant prise selon le résultat du test.

On trouve aussi parfois les symboles et qui indiquent que l'organigramme a une suit qui se trouve sur la page dont le numéro est indiqué dans le symbole (ex.: 2) ou bien que l'organigramme et la suite de l'organigramme se trouvent sur une autre page (ex.: 1).

Nous possédons maintenant les moyens de dessiner l'organigramme du jeu du 21 dont l'analyse a été faite précédemment.

<u>Organigramme principal du jeu du 21</u>



Il s'agit maintenant de dessiner les organigrammes secondaires correspondant aux différents modules que l'organigramme principal met en évidence.

La phase d'initialisation ne pose pas de problème. Il suffit de donner à une variable, que l'on pourra appeler AVOIR, la valeur initiale 1000.

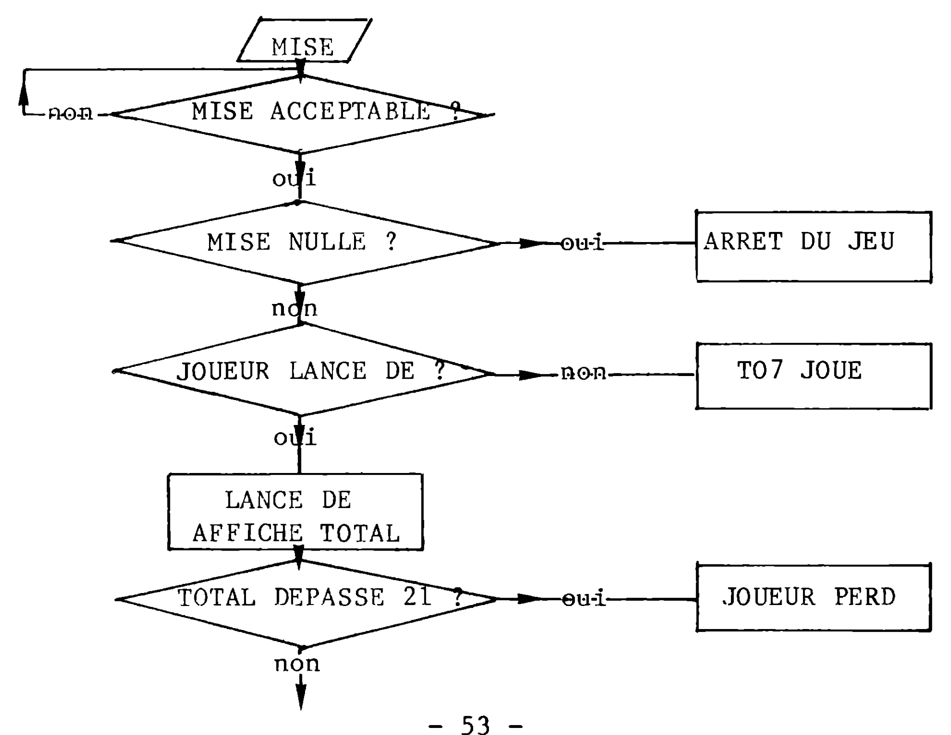
* ORGANIGRAMME DE LA PHASE DE JEU DU JOUEUR

. Le joueur doit d'abord miser. Sa mise ne doit pas dépasser son avoir. Si la mise est nulle, le jeu s'arrête.

. Le joueur peut ensuite soit lancer le dé soit laisser le TO7 jouer.

. Si le joueur lance le dé et obtient un total dépassant 21, le joueur a perdu. Si le total est inférieur à 21, le joueur se voit proposer un nouveau lancé.

L'organigramme est donc :



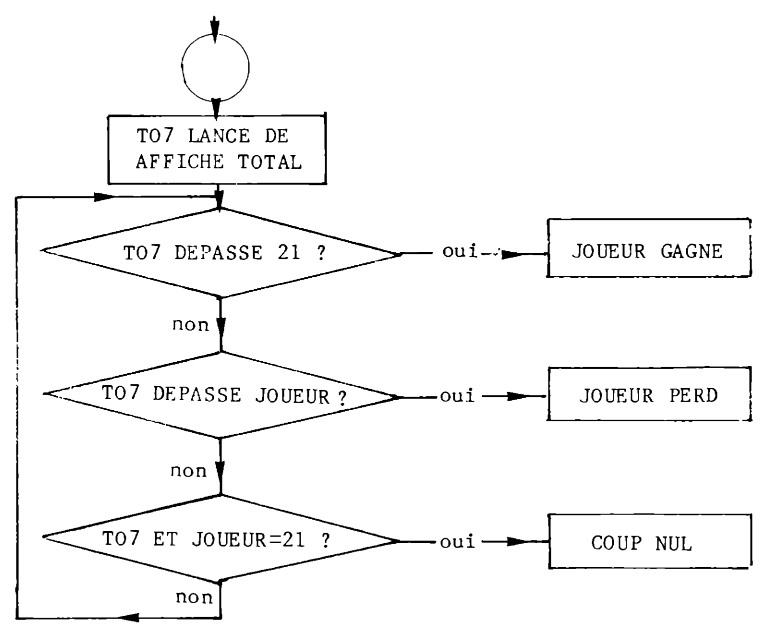
*   ORGANIGRAMME DE LA PHASE DE JEU DU TO7

. Le TO7 lance le dé.

. Si son total dépasse 21, le joueur a gagné.

. Si son total dépasse le total du joueur sans dépasser 21, le joueur a perdu.

. L'organigramme est donc :



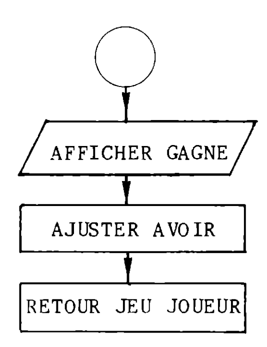
*   ORGANIGRAMME DE LA PHASE DE TRAITEMENT DES RESULTATS

. L'analyse et les organigrammes ci-dessus montrent que deux cas se présentent : le joueur gagne ou le joueur perd. L'avoir du joueur est ajusté en conséquence.

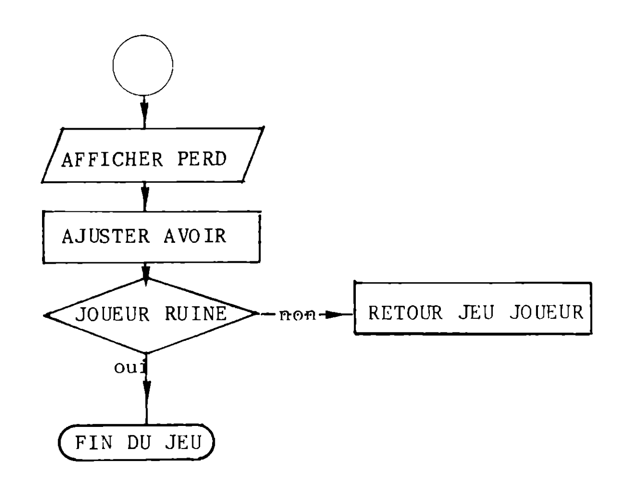
. Si le joueur est ruiné le jeu s'arrête, sinon le joueur a la possibilité de rejouer.

. On peut donc dessiner deux petits organigrammes, l'un pour JOUEUR GAGNE, l'autre pour JOUEUR PERD.

. JOUEUR GAGNE



. JOUEUR PERD



La décomposition est maintenant suffisante pour que l'on puisse passer à la phase de codage.

2.5.3 Codage

* Commençons dans un premier temps par écrire les instructions qui effectuent les initialisations nécessaires.

```
190   REM TO7 LANCE LE DE
200   MICRO = MICRO + INT(RND * 6 + 1)
210   PRINT "TO7 :    " ; MICRO
220   IF MICRO > 21 GOTO JOUEUR GAGNE
230   IF MICRO > JOUEUR GOTO JOUEUR PERD
240   IF MICRO + JOUEUR — 42 = 0 GOTO COUP NUL
250   GOTO 190
```

* Il ne reste plus qu'à coder les phases JOUEUR GAGNE, JOUEUR PERD et COUP NUL.

```
260   REM JOUEUR GAGNE
270   PRINT "VOUS GAGNEZ !!!"
280   AVOIR = AVOIR + INT(RND * 5 + 1) * MISE
290   GOTO 60
300   REM JOUEUR PERD
310   PRINT "VOUS PERDEZ..."
320   AVOIR = AVOIR - MISE
330   IF AVOIR > 0 GOTO 60
340   PRINT "VOUS ETES RUINE !!!"
350   END
360   REM COUP NUL
370   PRINT "COUP NUL !!!"
380   GOTO 60
```

* Il faut maintenant remplacer les étiquettes que nous avons placées tout au long du programme par les numéros de lignes correspondants.

La ligne 120 devient :

```
120   IF MISE = 0 GOTO 350
```

La ligne 140 devient :

```
140   IF J$ <> "" GOTO 190
```

La ligne 170 devient :

```
170  IF JOUEUR > 21 GOTO 300
```

La ligne 220 devient :

```
220  IF MICRO > 21 GOTO 260
```

La ligne 230 devient :

```
230  IF MICRO > JOUEUR GOTO 300
```

La ligne 240 devient :

```
240  IF MICRO + JOUEUR - 42 = 0 GOTO 360
```

* Vous pouvez dès à présent utiliser ce programme pour jouer.

Entrez votre mise et pressez [ENTREE]. Si votre mise est égale à zéro, le jeu s'arrête.

Pour lancer le dé, pressez simplement [ENTREE].

Lorsque vous estimez votre total suffisant, pressez une touche quelconque puis [ENTREE]. Le TO7 lance alors le dé à son tour.

2.5.4 Amélioration du jeu du 21

Plusieurs modifications peuvent être faites sur le programme précédent pour en rendre la présentation plus attrayante.

Profitons du fait que les lignes sont espacées de dix en dix, nous allons pouvoir insérer de nouvelles lignes intervenant essentiellement sur l'affichage.

```
10  REM
20  REM JEU DU 21
30  REM
40  REM INITIALISATIONS
50  AVOIR = 1000
60  MICRO = 0
70  JOUEUR = 0
```

. Nous mettons volontairement beaucoup de REM car un programme très commenté et aéré est beaucoup plus lisible.

. La variable AVOIR, qui contient le crédit du joueur, est initialisée à 1000 au début du jeu.

. Les variables MICRO et JOUEUR sont les totaux réalisés en lançant le dé par le TO7 et le joueur. Ces totaux sont évidemment nuls au début du jeu.

Remarquons tout de suite que ces totaux doivent être remis à zéro entre chaque tour de jeu.

* Ecrivons maintenant les instructions permettant d'exécuter les phases de jeu du joueur; regardez en même temps, si possible, l'organigramme correspondant.

```
 80  REM JOUEUR LANCE LE DE
 90  PRINT "AVOIR : " ; AVOIR
100  INPUT "MISE" ; MISE
110  IF MISE > AVOIR GOTO 100
120  IF MISE = 0 GOTO ARRET DU JEU
```

. Nous voyons que la ligne 120 pose un problème : l'organigramme exige, dans le cas où la mise est nulle, de se brancher sur ARRET DU JEU.

Cette partie du programme n'est pas encore écrite, nous ne savons donc pas quelle est pour l'instant le numéro de ligne à mettre après l'instruction GOTO. Mettons donc pour l'instant l'"étiquette" ARRET DU JEU. Nous pourrons remplacer cette étiquette par un numéro de ligne quand le programme sera entièrement écrit.

De manière générale, chaque fois qu'un branchement sera exigé alors que nous ne connaissons pas le numéro de ligne correspondant, nous mettrons une étiquette. Les étiquettes seront remplacées par des numéros de ligne quand le programme sera terminé.

```
130  INPUT "JET" ; J$
140  IF J$<>"" GOTO TO7 JOUE
150  JOUEUR = JOUEUR  + INT(RND * 6 + 1)
160  PRINT "JOUEUR : _ " ; JOUEUR
170  IF JOUEUR >21 GOTO JOUEUR PERD
180  GOTO 130
```

. Jusqu'à présent nous n'avions utilisé que des instructions que vous connaissiez. La ligne 150 fait appel à deux fonctions nouvelles :

INT(X) donne la partie entière du nombre X, c'est-à-dire le plus grand entier inférieur ou égal à X.

RND donne un nombre réel aléatoire compris entre zéro et un, un n'étant jamais atteint.

INT(RND * 6 + 1) donne donc un nombre entier compris entre 1 et 6, simulant ainsi le lancer d'un dé à six faces.

* Passons maintenant à la phase de jeu du TO7.

Nous vous donnons tout de suite le "listing" du programme modifié. Vous trouverez ensuite des explications détaillées sur chacune des modifications apportées.

Vous pouvez effectuer directement sur le programme précédent les modifications, celles-ci n'étant que des insertions de ligne ou d'instructions sur des lignes déjà existantes. Il ne s'agit donc pas d'un nouveau programme mais du même programme "étoffé".

JEU DU 21

```
 10  REM
 20  REM JEU DU 21
 30  REM
 40  REM INITIALISATIONS
 42  CLS
 44  CLEAR ,, 2
 45  GOSUB 500
 50  AVOIR = 1000
 60  GOSUB 700
 65  CLS : MICRO = 0
 70  JOUEUR = 0
 80  REM JOUEUR LANCE LE DE
 90  PRINT "AVOIR : " ; AVOIR : PRINT
100  INPUT "MISE" ; MISE
110  IF MISE > AVOIR GOTO 100
120  IF MISE = 0 GOTO 350
130  PRINT : INPUT "JET" ; J$
140  IF J$<>"" GOTO 190
150  JOUEUR = JOUEUR  + INT(RND * 6 + 1)
160  PRINT "JOUEUR : _ " ; JOUEUR
170  IF JOUEUR > 21 GOTO 300
180  GOTO 130
190  REM TO7 LANCE LE DE
200  MICRO = MICRO + INT(RND * 6 + 1)
205  GOSUB 700
210  PRINT : PRINT "TO7: _"; MICRO
220  IF MICRO > 21 GOTO 260
230  IF MICRO > JOUEUR GOTO 300
```

```
240  IF MICRO + JOUEUR - 42 = 0 GOTO 360
250  GOTO 190
260  REM JOUEUR GAGNE
270  PRINT : PRINT "VOUS GAGNEZ!!!"
280  AVOIR = AVOIR + INT(RND * 5 + 1) * MISE
290  GOTO 60
300  REM JOUEUR PERD
310  PRINT : PRINT "VOUS PERDEZ..."
320  AVOIR = AVOIR - MISE
330  IF AVOIR > 0 GOTO 60
340  PRINT "VOUS ETES RUINE !!!"
350  PRINT : INPUT "VOULEZ VOUS REJOUER_" ;
     R$
352  IF R$ = "" GOTO 10
355  END
360  REM COUP NUL
370  PRINT "COUP NUL !!:"
380  GOTO 60
500  REM AFFICHAGE
510  SCREEN 3, 0, 0
520  ATTRB 1,1
530  LOCATE 10, 14, 0 : PRINT "JEU DU 21"
540  DEFGR$ (0) = 255, 129, 165, 129, 129,
     165, 129, 255
550  DEFFR$(1) = 255, 129, 161, 129, 129,
     133, 129, 255
560  LOCATE 5, 5, 0 : COLOR 1 : PRINT GR$(0)
570  LOCATE 35, 20, 0 : COLOR 5 : PRINT
     GR$(1)
580 ATTRB 0, 0
590  LOCATE 0, 24, 1 : COLOR 2 : INPUT
     "ENTREZ UN NOMBRE QUELCONQUE" , N
600  N = INT(ABS(N))
610  FOR I = 1 TO N
620  Y = RND
630  NEXT I
640  RETURN
700  REM TEMPORISATION
710  FOR I = 1 TO 500
720  NEXT I
730  RETURN
```

. L'instruction CLS, que l'on trouve dans les lignes 42 et 65, a pour but d'effacer l'écran du téléviseur et de positionner le curseur dans le coin supérieur gauche de l'écran.

. L'instruction CLEAR ,, 2 permet de réserver de la place mémoire pour la définition de deux caractères graphiques.

. De nombreux PRINT ont été ajoutés (lignes 90, 130, 210, 270, 310, 350) afin d'aérer la présentation du jeu.

. Un sous-programme d'affichage, apporté par la ligne 45, est placé à partir de la ligne 500. Ce sous-programme fait appel à beaucoup d'instructions particulières du BASIC du TO7. Ces instructions permettent d'exploiter les très bonnes possibilités graphiques du TO7 :

- définition de caractères graphiques : DEFGR$

- couleurs de l'écran et des caractères : SCREEN et COLOR.

- taille des caractères ATTRB.

- positionnnement du curseur : LOCATE.

Vous trouverez les détails complets sur ces instructions et leurs syntaxes particulières dans le chapitre 4.

Une instruction INPUT vous demande d'entrer un nombre quelconque. Cela a pour effet de faire "tourner" un certain nombre de fois le générateur aléatoire afin d'obtenir dans la suite du jeu des séquences différentes.

. Un sous-programme de temporisation est placé à partir de la ligne 700. Il s'agit simplement d'une boucle vide parcourue 500 fois.

. Ce sous-programme est appelé par les lignes 60 et 205. Il est en effet nécessaire de ralentir artificiellement le jeu sinon cela va trop vite pour le joueur : affichage trop fugitif, tirage de dé trop rapide pour le TO7.

. Vous venez d'assister à la création complète d'un jeu en BASIC. Vous avez pu constater que cela n'est vraiment pas difficile si on procède avec méthode.

Le jeu que nous vous avons proposé n'est certainement pas parfait. Nous espérons que vous aurez des idées pour l'améliorer et que vous n'hésiterez pas à les mettre en application.

Nous souhaitons que tout ce qui précède vous aura donné envie de programmer vous-même vos propres jeux, tel en tout cas était notre but.

CHAPITRE **3**

Le BASIC du TO 7

3.1 INTRODUCTION

Vous voilà maintenant un peu familiarisé avec votre Thomson TO7. Poursuivons avec l'étude, en détail, du BASIC du TO7.

Cette étude se compose de quatre parties, à savoir :

1) Description des "éléments"
2) Différentes catégories d'instruction BASIC
3) Notions de sous-programmes et de fonctions
4) Entrées-sorties généralisées

3.2 DESCRIPTION DES ÉLÉMENTS

Qu'entend-on par "éléments" ? Dans tout programme on est amené à manipuler : des matrices, des nombres entiers, réels etc... et/ou des chaînes de caractères sur lesquels on effectue des opérations ; toutes ces données constituent "les éléments". Parmi eux on distingue les opérandes et les opérateurs. Ainsi, afin de décrire les éléments, on procèdera en deux temps.

- Description des opérandes

- Description des opérateurs

3.2.1 Description des opérandes

Parmi ceux-ci il y a deux types ; d'une part les constantes, d'autre part les variables. La différence entre ces deux types ayant été décrite dans le chapitre précédent (cf 2.4.1), on rappelera simplement pour mémoire qu'il y a :

- les constantes

  . numériques qui sont entières*, réelles, double précision, octales, hexadécimales ou binaires
  . chaînes de caractères

- les variables

  . numériques qui sont entières*, réelles ou double précision
  . chaînes de caractères
  . matricielles (aussi appelées tableaux)
  (Il convient de noter que les matrices sont variables à double titre :
  - par leur contenu
  - par leur structure
  Ceci sera d'ailleurs explicité ultérieurement).

Les divers opérandes vont être explicités en détail.

---

* "entier" en langage informatique correspond à un entier relatif (ou rationnel) au sens strictement mathématique.

* CONSTANTES ET VARIABLES ENTIERES

La notion (mathématique) de nombre entier s'identifie au comptage d'objets (ex. : nombre de poules dans une basse-cour). En informatique, cette notion est "généralisée" aux "entiers négatifs" (ex. : un compte bancaire présentant un découvert de 1300 francs est considéré comme étant à -1300 francs (moins 1300 francs)). Le TO7 accepte les entiers compris entre -32768 et 32767 inclus, soit :

-32768, -32767, -32766, ..., -2, -1, 0, 1, 2, ..., 32766, 32767

Les variables entières (ou les cases mémoires réservées aux nombres entiers) sont repérées par l'ordinateur par le suffixe % (qui se trouve ajouté à leur nom.

Exemple :

A% désigne une variable entière,
A ne désigne pas une variable entière.

Remarques : 1. De nombreux ordinateurs codent les entiers sur deux octets, soit 16 bits. Un bit sert à déterminer le signe du nombre (positif ou négatif), les 15 autres déterminent sa valeur, ce qui explique les valeurs limites (2 ↑ 15 = 32768).

2. Si l'utilisateur désire travailler avec des nombres entiers plus importants, il faut qu'il utilise les réels associés (cf ci-après).

* CONSTANTES ET VARIABLES REELLES

Les nombres réels sont utiles dès qu'il faut diviser les nombres (au sens strictement mathématique, cette démarche n'est pas exacte, il y aurait lieu de raisonner en termes de racine carrée au lieu de division). Il est important de retenir qu'il existe des opérateurs qui, appliqués à des entiers, conduisent à des résultats non entiers, à savoir les réels (ex. : 1/2 ne correspond pas à un entier, il est impossible de diviser une voiture en deux parties sans que celle-ci perde son caractère auto-mobile). Néanmoins dans beaucoup de cas il faut réaliser ce type de division (ex. : l'achat d'un demi-kilog de carottes à 5 francs coûte 5/2 francs). Pour représenter le résultat de telles divisions, il faut recourir aux nombres réels. Intuitivement, il s'agit des nombres possédant (ou pouvant posséder) une partie décimale. Le TO7 accepte des nombres réels entre

1 EE 35 (soit 100.000......000) (jusqu'à 35 zéros) et -1 EE 35 (idem, précédé d'un signe moins, "-"),

possédant jusqu'à 5 chiffres décimaux ; au-delà, il faut utiliser des nombres double précision (cf ci-après). Une place mémoire réservée à un réel (ou variable réelle) est repéré par le TO7 par le suffixe ! ajouté à son nom. Il est à préciser que ce suffixe est optionnel (en pratique, il est toujours omis, ce qui n'estpas un inconvénient, car l'ordinateur considère d'autorité que les noms de variables ne se terminant pas pas %, # ou $ sont des réels.

Exemples :

A est une variable réelle
Score est une variable réelle
1.56 est une constante réelle
1.5678675 n'est pas pour le TO7 une constante réelle (il y a 7 chiffres décimaux)

Remarque : les nombres réels sont codés (par le TO7) sur 4 octets.

* CONSTANTES ET VARIABLES DOUBLE PRECISION

Lorsqu'on fait des calculs, parfois, on souhaiterait obtenir de la part de la machine plus de chiffres significatifs. Ainsi, pour la plupart des micro-ordinateurs, si rien n'est précisé de la part de l'utilisateur, les résultats sont fournis avec 6 ou 7 chiffres significatifs. Mais s'il est précisé à l'ordinateur qu'il doit en fournir davantage, les résultats apparaîtront avec 15 chiffres (significatifs) après la virgule.

Cet accroissement de la précision est simplement dû au fait qu'on a substitué aux variables et aux constantes que l'on utilisait des variables et des constantes dites "double précision". Pour indiquer au TO7 que les variables et les constantes sont de type "double précision", il suffit d'ajouter le suffixe # aux noms des variables et des constantes. Pour vous familiariser avec ceci vous pouvez taper le programme suivant :

```
10  INPUT "DONNEZ UN NOMBRE" ; A#
20  PRINT A#
30  A = A#          (cette ligne sera justifiée
40  PRINT A          par la suite)
```

Si vous le faites exécuter, le message suivant est imprimé :

"DONNEZ UN NOMBRE ?"

En frappant par exemple :

1.12345678901234567 [ENTREE]

l'ordinateur imprime :

1.123456789012346
1.12346

OK

Remarque : il apparaît clairement que le type "double précision" ne concerne que des valeurs numériques (et ne s'applique pas aux chaînes de caractères).

* CONSTANTES BINAIRES OCTALES ET HEXADECIMALES

En supplément de la base 10 habituellement utilisée le TO7 accepte des constantes (et non des variables) exprimées en base 2, 8 ou 16.

Remarque : Les ordinateurs calculent en base deux. Tout nombre est exprimé avec deux chiffres 0 et 1. Les humains ont l'habitude de calculer en base 10 (tout nombre est exprimé avec 10 chiffres 0, 1, 2, ..., 9).

Il existe une relation entre les différentes bases qui permet de transcrire tout calcul exprimé dans une base dans une autre base. Pour être prise en compte par le TO7 toute constante exprimée dans une base autre que la base 10 doit être précédée des symboles respectifs suivants :

&B pour la base 2 (B pour binaire)

&O (ou plus simplement &) pour la base 8 (O pour octale)

&H pour la base 16 (H pour hexadécimale)

Exemples :

&B 101 désigne une constante binaire

&B 20 ne correspond à rien car en binaire il n'y a que les chiffres 0 et 1.

Enfin, dans tous calculs ou affectations (cf ci-après) les constantes exprimées dans une base différente de la base 10 sont restituées en base 10 au niveau des organes de sortie (de l'ordinateur) : écran, imprimante etc...

Exemples : Avec les instructions :

```
PRINT &B101 [ENTREE]
```

le TO7 imprime :

```
5
OK
```

(En effet 5 en base 10 vaut 101 en base 2)

Avec :

```
PRINT &B20 [ENTREE]
```

le TO7 imprime :

```
? SN Error
OK
```

Les différents types de variables abordés jusqu'à présent correspondent aux types fondamentaux ou de base. On peut les résumer dans le tableau suivant.

Remarque : la structure des noms de variables a été décrite en détail dans le chapitre précédent. Ainsi dans le tableau ci-dessous AB a un caractère symbolique et désigne un nom de variable.

TYPE	NOM DE VARIABLE	VALEURS EXTREMES
Entier	AB %	-32768 ; +32767
Réel	AB ! (ou AB)	± 9.99999 E +35
Double précision	AB #	± 9.9999999999999999D+35
Chaîne de caractères	AB$	de 0 à 255 caractères

* VARIABLES MATRICIELLES
(synonymes : matrices, tableaux)

Une matrice permet de regrouper un ensemble de variables de même type sous un nom commun. Ainsi il existe des matrices d'entiers, de réels, de variables "double précision" ou de chaînes de caractères. Le nom commun sous lequel est regroupé l'ensemble des variables est le nom de la matrice. En BASIC il répond aux mêmes restrictions que celui d'une variable de base (cf 2.4.1).

Exemples:

AB% désigne une matrice d'entiers
AB$ désigne une matrice de chaînes de caractères

Remarque : AB a un caractère symbolique.

Pour rentrer ou ressortir les variables placées dans la matrice, il est nécessaire de désigner les emplacements dans lesquels elles se trouvent. On ne peut pas mettre en vrac des variables dans un tableau, sans quoi on ne saurait plus les distinguer les unes des autres. Ainsi on localise une variable au moyen d'indices. Le nombre d'indices nécessaires pour localiser une variable définit la dimension de la matrice. A titre d'exemple, des entiers ont été placés dans une matrice à deux dimensions (cf figure 1). Dans celle-ci le nombre 8 est en deuxième ligne et quatrième colonne. Les indices de la case où se trouve 8 sont donc 2, 4.

Figure 1.

1					
			8	18	
25					

Remarques :

. Il ne faut pas confondre 2, 4 et 4, 2. En effet 4, 2 repèrent la case où figure 25.

. Une case peut être "vide". La lecture de son contenu se traduit dans certains cas par l'affichage d'un message d'erreur (sur le TO7 ; en général la lecture du contenu d'une case "vide" donne 0).

. Un indice est toujours un nombre entier positif (ou nul). Si on désigne une case par des indices :

- non entiers l'ordinateur les transforme en les indices entiers les plus proches.
- négatifs l'ordinateur affiche le message d'erreur :

? FC Error

OK

Sur le TO7 on peut utiliser des tableaux dont la taille ou dimension est variable jusqu'à 255. En fait, en pratique, elle dépend du volume "mémoire" disponible. Si pour une matrice, il y a plus de trois dimensions ou des indices supérieurs à 10, il faut "prévenir" le TO7. Ceci se fait au moyen d'une déclaration de matrice(s). Elle se fait :

- dans l'écriture du programme

- en principe au début ; du moins à un endroit du programme tel que lors de l'exécution la déclaration soit prise en compte avant toute opération avec les tableaux concernés.

La syntaxe d'une déclaration est donnée en détail dans le manuel de référence du TO7, elle est rappelée ci-après :

DIM /matrice/ [, /matrice/...]

avec

/matrice/ = /nom de variable/ (indice [,indice]...)

Exemple :

```
10 DIM  A% (10), B$ (2,5)
```

déclare une matrice

- A% d'entiers à une dimension dont l'indice varie de 0 à 10

et (une matrice)

- B$ de chaînes de caractères à deux dimensions dont les indices varient de 0 à 2 et de 0 à 5.

Remarques :

1) Si un indice dépasse la valeur maximale qui lui est permise le TO7 affiche :

```
? BS Error

OK
```

Exemple : reprendre la ligne 10 de l'exemple ci-dessus avec la ligne 20 suivante :

```
20 PRINT A% (11)
```

Ce programme donne à l'exécution

```
? BS Error In 20
```

(20 est le numéro de la ligne où est apparue l'erreur)

L'erreur est due au fait que le tableau A % ne contient que 11 cases (de 0 à 10). Par conséquent la case numéro 11 n'existe pas.

2) Chaque tableau déclaré a une dimension fixée (après la prise en compte de la déclaration). De ce fait pour tout travail sur les tableaux (déclarés), le nombre d'indices devra toujours être égal à la dimension du tableau.

Exemple :

A l'exécution du programme suivant

```
10  DIM B$ (2, 3. 10)
20  PRINT B$ (1, 2, 8, 5)
```

L'ordinateur affiche le message suivant :

```
? BS Error In 20
```

3) La plage de variation d'un indice (ou de plusieurs indices) peut être rendue variable par l'utilisation d'une (ou de plusieurs) variable(s) auxiliaire(s).

Exemple :

```
10  INPUT "PLAGES DE VARIATION" ; N ; P ; Q
20  DIM A (N, P, Q, 10)
```

Par ces deux instructions on déclare un tableau A ayant 4 dimensions dont les indices peuvent varier respectivement de 0 à N, de 0 à P, de 0 à Q et de 0 à 10. Les valeurs de N, P, Q sont à fixer lors de l'exécution du programme.

Note : cette possibilité (de rendre variables les plages de variation) est intéressante pour un jeu de bataille navale. Elle permet de disposer d'un océan de taille variable.

*   CONVERSION DE TYPE

Elle est spécifique des variables numériques (les chaînes de caractères ne sont pas concernées). En quoi consiste-t-elle ? En principe toute opération, comme par exemple l'addition, ne peut être effectuée qu'entre des opérandes de même type numérique. C'est pourquoi le TO7 réalise automatiquement - si nécessaire - la conversion des types sur les opérandes.

Exemple :

En écrivant

```
A% = 5 : B = 8 [ENTREE]
PRINT A% + B [ENTREE]
```

Le TO7 affiche

```
13

OK
```

Pour ce faire, il convertit le nombre 5 entier figurant dans A% en le nombre 5 réel et peut ainsi réaliser l'addition avec le 8 réel figurant dans B.

D'autre part les noms des emplacements mémoire, dans lesquels se trouvent placées des valeurs, sont caractéristiques du type de variables qu'ils contiennent.

Exemple :

A$ est un emplacement mémoire ne contenant que des chaînes de caractères.

B% ne contient que des entiers.

Ainsi, lors de la copie de la valeur d'une mémoire dans une autre (ou lors du stockage d'une constante en mémoire), il se peut que le type de l'élément à recopier (ou à stocker) soit différent du type d'éléments contenus dans l'emplacement mémoire dans lequel il va figurer. Pour pouvoir faire la copie (ou le stockage), le TO7 procède automatiquement à une conversion de type.

Exemples :

A = A%	(un entier est copié dans une mémoire A réservée à des réels)
A% = 56.8	(un nombre non entier doit être stocké dans une mémoire ne pouvant contenir que des entiers. Dans ce cas le TO7 va transformer 56.8 en l'entier le plus proche (soit 57) et stocker celui-ci).

Le principe de conversion de type automatique caractérise la souplesse d'utilisation du BASIC. L'utilisateur (de BASIC) n'a pas à s'en soucier (car il se fait automatiquement) mais il est utile qu'il en soit conscient. En effet, dans d'autres langages, tels que le PASCAL, celle-ci ne se fait pas automatiquement et cela pose quelquefois des problèmes.

* DEFINITION IMPLICITE DU TYPE DE VARIABLE

DEFINT/DEFSNG/DEFDBL/DEFSTR

```
Syntaxe :   ( DEFINT ) (                              )
            ( DEFSNG ) ( /lettre/ [-/lettre/] [,] )
            ( DEFDBL ) (                              )
            ( DEFSTR ) (                              )
```

Ces quatre instructions sont pratiques dans la mesure où beaucoup de variables de même(s) type(s) sont utilisées dans un programme. Elles permettent respectivement de définir les variables dont la première lettre (de leur nom) fait partie d'un certain ensemble comme étant de type :

- entier
- réel
- double précision
- chaîne de caractères

Ceci permet d'omettre les suffixes %, !, #, $

Exemple :

```
DEFINT A-B, O-Q
```

permet de lire les variables dont le nom commence par A, B,O,P ou Q comme étant de type entier.

3.2.2 Description des opérateurs

Il y a quatre types d'opérateurs, à savoir opérateurs numériques, opérateurs sur chaînes de caractères, opérateurs relationnels et opérateurs logiques.

Ils sont abordés successivement (dans l'ordre ci-dessus).

* OPERATEURS NUMERIQUES

Ces opérateurs sont très connus, ils sont rappelés dans l'ordre de priorité décroissante.

( ) : parenthèses qui permettent de délimiter des expressions.

- : opérateur unaire qui permet de changer le signe d'une constante. Exemple : -3

↑ : élévation à la puissance (X ↑ Y)

* / : multiplication (*) et division (/)

@ MOD : division entière (@) et modulo

+ - : addition et soustraction

La division entière et le "modulo" sont des opérations qui nécessitent deux opérandes (exemple: 3@ 2 ; 5 MOD 2). Elles donnent respectivement, après avoir arrondi les opérandes (si nécessaire), le quotient et le reste de la division.

Exemples :

3@ 2 donne 1 car 3 = 1 * 2 + 1

5 MOD 2 donne 1 car 5 = 2 * 2 + 1

Remarques :

. Toute division par 0 entraîne l'affichage

? / 0 Error

. Un dépassement de capacité de l'ordinateur avec des nombres trop grands est signalé par le message

? OV Error

Exemple :

En entrant

PRINT 1 EE + 35 ↑ 2 [ENTREE]

on obtient

? OV Error

OK

* OPERATEURS SUR CHAINES DE CARACTERES

Il y a un seul opérateur appelé la concaténation. Il exige deux opérandes (deux chaînes de caractères) qu'il accolle. Son symbole est + (comme pour l'addition).

Exemple :

PRINT "BONJOUR_" + "MONSIEUR" [ENTREE]

donne

BONJOUR_MONSIEUR

OK

Remarque : le symbole _ représente un "espacement" (ou caractère blanc).

* OPERATEURS RELATIONNELS

Ils servent à faire des tests et seront très utiles dans tout programme. Ils sont donc très importants. Ils exigent tous deux opérandes, qui sont soit deux nombres (entiers, réels etc...), soit deux chaînes de caractères. Le résultat d'une opération (relationnelle) ne peut prendre que deux valeurs : soit "vrai" soit "faux". (Le TO7 représente "vrai" par le chiffre - 1 et "faux" par 0).

Ces opérateurs ont tous la même priorité et elle est inférieure à celle de tout opérateur numérique. Les différents opérateurs sont :

>	"strictement supérieur à"
<	"strictement inférieur à"
>= (ou =>)	"supérieur ou égal à"
<= (ou =<)	"inférieur ou égal à"
>< (ou<>)	"différent de"
=	"égal à"

Exemple :

Souvent les programmes de jeux utilisent deux mémoires (spécifiques), appelées SCOREA et SCOREB, qui contiennent les scores des joueurs, A et B. A la fin du programme il existe une instruction qui permet de déterminer le gagnant. Cette instruction utilise un opérateur relationnel.

Les manières d'utiliser ces opérateurs sont très variées.

a) D'une manière générale, du fait que le résultat d'une "comparaison" est un nombre 0 ou - 1, les opérateurs relationnels (ou "comparaison") peuvent être introduits dans les suites numériques.

Exemple :

```
10  INPUT ; A, B
20  PRINT A = B
30  C = 3 * (A = B) + (A < B)
40  PRINT "C =" ; C
```

Les instructions ci-dessus sont correctes.

Cette possibilité d'inclure dans les expressions numériques des opérations relationnelles est très importante. Le lecteur s'en rendra compte en programmant. Cependant il faut faire attention à la mise entre parenthèses, sans quoi l'on risque de ne pas être en accord avec le but fixé.

Exemple :

```
3 * (A = B)
```

ne donne pas le même résultat que

```
3 * A = B
```

En effet l'instruction 3 * (A = B) vaut - 3 si A est égal à B et 0 sinon. Par contre 3 * A = B prendra la valeur - 1 si A est égal au tiers de B et 0 sinon.

b) On peut combiner des opérations relationnelles au moyen d'opérateurs logiques (cf ci-après).

c) Les opérateurs relationnels sont indispensables pour les instructions de branchements conditionnels (cf ci-après).

* OPERATEURS LOGIQUES

Ces opérateurs, sauf un, exigent deux opérandes qui sont en théorie des booléens, et le résultat (d'une opération logique) est un booléen. Un booléen peut prendre par définition deux valeurs : soit la valeur "vrai" (qui est représentée par - 1 sur le TO7), soit la valeur "faux" (qui est représentée par 0 sur le TO7).

Exemples de booléens :

La proposition "un chat est un chien" est un booléen qui a la valeur "faux".

La proposition "2 + 2 = 4" (en base 10) est un booléen qui a la valeur "vrai".

Les différents opérateurs sont :

AND appelé "et logique"

EQV appelé "équivalent logique"

IMP appelé "implication logique"

OR appelé "ou logique"

XOR appelé "ou exclusif logique"

NOT appelé "négation" (c'est le seul opérateur exigeant un seul opérande)

Ils sont entièrement définis par le tableau ci-dessous (cf figure 2).

1er booléen (A)	2ème booléen (B)	A AND B	A EQV B	A IMP B	A OR B	A XOR B
0	0	0	- 1	- 1	0	0
0	- 1	0	0	- 1	- 1	- 1
- 1	0	0	0	0	- 1	- 1
- 1	- 1	- 1	- 1	- 1	- 1	0

et

booléen (A)	NOT A
0	- 1
- 1	0

Figure 2

Ces opérateurs servent essentiellement à combiner des opérations relationnelles.

Exemple :

(A = B) AND (B > C) est une expression utilisant la combinaison de deux opérations relationnelles.

Néanmoins avec le TO7 ces opérations sont "extensibles" à des valeurs numériques. Pour comprendre ces possibilités il faut connaître "la représentation des nombres".

Qu'est-ce que la représentation des nombres ?

L'ordinateur ne travaille qu'avec les chiffres 0 et 1. De ce fait tout ce qui est entré par le clavier est converti en une suite de 0 et de 1 (pour que l'ordinateur puisse comprendre). En particulier les nombres entiers sont convertis (en suites de 0 et de 1). Les suites nécessaires pour coder les entiers utilisent 16 chiffres (0 ou 1). Il est possible de vérifier que si le codage binaire d'un nombre entier se termine par :

un 0 le nombre entier (codé) est divisible par 2 (= 2↑1)

deux 0 le nombre est divisible par 4 (= 2↑2)

trois 0 le nombre est divisible par 8 (= 2↑3)

un 1 le nombre est impair etc...

Ainsi le codage binaire (qui est la représentation des nombres) fournit des renseignements sur la parité et la divisibilité des nombres entiers.

Par suite les instructions suivantes

- Y = X AND &B1111111111111110 (X désigne un entier).

- Z = X AND &B1111111111111100

- U = X OR &B0000000000000001

auront pour effets respectifs de :

- remplacer le dernier chiffre du codage binaire de X par un 0. De ce fait Y est le plus grand nombre entier inférieur à X divisible par 2.

- remplacer les deux derniers chiffres du codage de X par deux 0. Z est donc le plus grand entier inférieur à X divisible par 4.

- remplacer le dernier chiffre du codage de X par un 1. U est donc le plus petit entier supérieur à X impair.

3.3 DIFFÉRENTES CATÉGORIES D'INSTRUCTIONS BASIC

Après avoir, si besoin est, établi un organigramme aussi détaillé que possible, il faut le traduire en une suite d'instructions compréhensibles par l'ordinateur (afin de rendre le programme exécutable). Cette traduction nous l'avons vu, nécessite la connaissance de ce qui est appelé un langage. Dans le cas présent il s'agit du BASIC Microsoft version 1.0 (contenu dans la cassette Memo 7.

Parmi ces instructions on distingue :

- les instructions d'assignation
- les instructions de contrôle, de branchement
- les instructions de bouclage (ou les boucles)
- les instructions d'entrée-sortie

etc...

Dans les paragraphes suivants la syntaxe de chaque instruction est rappelée brièvement pour faire place à une plus ample explication. Pour chaque syntaxe ce qui est entre parenthèses ( ) est optionnel ; les termes entre crochets [ ] sont à remplacer par les valeurs appropriées (ex : [variable] peut être remplacé par SCORE A%). Enfin, pour les termes superposés entre accolades, un seul doit être conservé à chaque utilisation de l'instruction. Si une instruction est introduite avec une mauvaise syntaxe, le TO7 affiche le message d'erreur suivant :

? SN Error In X

où X est le numéro de la ligne du programme dans laquelle figure l'instruction erronée.

3.3.1 Instructions d'assignation

Il s'agit de mettre une valeur dans une variable. Deux cas peuvent se présenter :

- la variable n'a encore jamais été utilisée

- la variable a déjà été utilisée, ou bien il s'agit d'un tableau encore inutilisé mais déclaré (DIM).

Dans le premier cas, il n'y a qu'une possibilité d'assignation. Elle utilise l'instruction :

LET
(syntaxe LET [nom de variable] = [expression])

Cela signifie que l'expression évaluée va être placée dans la mémoire correspondant à la variable.

Remarque : le terme LET peut être omis. En pratique il l'est toujours.

Dans le deuxième cas il y a une possibilité supplémentaire. Avant de l'étudier il est nécessaire d'examiner la manière dont le TO7 mémorise les valeurs des variables.

Un ordinateur comporte un ensemble de mémoires dont une partie est réservée à l'utilisateur. Toutes ces mémoires sont "numérotées" (pour la partie utilisateur, la numérotation va de 24832 à 57343 inclus).

Dans cette partie se situe tout ce que vous créez : variables, programmes. Ainsi chaque variable, chaque instruction est rangée dans une ou plusieurs cases mémoire, selon la taille (de la variable, de l'instruction). De ce fait, pour changer la valeur (d'une variable) il suffit de demander à l'ordinateur l'adresse de celle-ci, puis de mettre dans cette (ou ces) case(s) mémoire la nouvelle valeur. Pour procéder de cette manière à une assignation, il est nécessaire de disposer de plusieurs instructions. Trois sont utilisables, à savoir : VARPTR, PEEK, POKE.

* **VARPTR** (VARiable PoinTeuR)

Syntaxe VARPTR [nom de variable]

Elle donne l'adresse de la première case mémoire dans laquelle se trouve la variable. Comme il a été précisé auparavant, une variable peut occuper plusieurs cases mémoire. En fait le nombre de cases utilisées est lié au type de la variable :

- un entier occupe deux cases mémoire :

  . la première contient les bits de poids fort
  . la deuxième ceux de poids faible

- un réel occupe quatre cases mémoire :

Exposant	Bits de poids fort de la mantisse	Bits de poids moyen	Bits de poids faible
Première case	Deuxième case	Troisième case	Quatrième case
Exposant	Mantisse		

- un nombre double précision occupe huit cases mémoire.

La première contient l'exposant ; les sept autres contiennent la mantisse.

- une chaîne de caractères occupe un nombre de places dépendant de sa longueur (dans ce cas la première case contient la longueur de la chaîne).

En résumé une variable entière occupe les cases VARPTR (nom de variable) et VARPTR (nom de variable) + 1 etc...

* PEEK

Syntaxe PEEK (I) où I est un entier positif inférieur à 65 535 (à valeur entière).

Elle donne la valeur contenue dans la case mémoire numéro I.

Exemple :

```
10  INPUT "DONNEZ UN NOMBRE ENTIER" ; A%
20  ADR = VARPTR (A%)
30  PRINT PEEK (ADR) ; PEEK (ADR + 1)
40  END
```

Ce programme permet de visualiser la façon dont les entiers sont mémorisés. Il est possible de vérifier que dans la mémoire numéro

ADR il y a le <u>reste</u> de la division entière de A% par 256
ADR + 1 il y a le <u>quotient</u> de cette même division

Remarque : la valeur 256 résulte du codage sur 2 octets (1 octet représente 8 bits) des entiers. Ceci n'est pas fondamental dans une première approche.

* POKE

Syntaxe POKE I, J

Elle place dans la case mémoire numéro I la valeur J. Le nombre I doit être positif inférieur à 65535 ; quant à J, il doit être entier positif inférieur à 255 (une case mémoire contient 1 octet, soit 8 bits, soit 2↑8 ou 256 valeurs possibles, d'où la valeur maximale de J). Si les valeurs de I et J introduites ne sont pas correctes, le TO7 affiche:

? FC Error (FC pour Function Call)

OK

En utilisant VARPTR et POKE, il est possible de modifier le contenu d'une variable.

Exemple :

10 INPUT "NOMBRE" ; A%	
20 ADR = VARPTR (A%)	(ADR est l'adresse de A%)
30 POKE ADR, 0	(0 est placé dans la case mémoire ADR)
40 POKE ADR + 1, 0	(0 est placé dans la case mémoire ADR +1)
50 PRINT "A% =" ; A%	

Ce programme met 0 dans la variable A%. En effet à l'exécution si à la question de l'ordinateur :

NOMBRE ?

on répond

35 [ENTREE]

le résultat obtenu est:

A % = 0

OK

Remarque : Pour modifier :

- un réel il faut 4 instructions POKE car il y a 4 cases mémoire concernées ;

- un nombre double précision, il en faut 8 ;

- une chaîne de caractères, le nombre dépend de la longueur de la chaîne.

Dans un premier temps l'assimilation des notions précédentes n'est pas indispensable. Mais elle est importante pour approcher davantage le fonctionnement de l'ordinateur. Ceux qui sont intéressés par les parties "cachées" du TO7 (i.e. les parties mémoire qui ne sont pas destinées à l'utilisateur) peuvent mettre à profit PEEK et POKE.

* CLEAR

Syntaxe CLEAR (I) ; (J) ; (K).

Remarque préliminaire : I, J, K sont des valeurs numériques en théorie entières et optionnelles (une au moins doit être présente). Elles seront arrondies si nécessaire par le TO7. Par contre, si elles sont négatives ou si K est supérieur à 128, une erreur de syntaxe est détectée (? SN Error).

Ses fonctions sont multiples :

- La présence de I ordonne au TO7 de réserver I octets ou I "cases mémoires" en place mémoire pour manipuler des chaînes de caractères. Il est conseillé de l'utiliser dès qu'un programme aura à effectuer la concaténation de chaînes (longues).

- La présence de J interdit au TO7 d'utiliser les cases mémoires réservées à l'utilisateur (cf POKE, PEEK, VARPTR) au-delà de la case mémoire de numéro J. Ceci est utile pour réserver de la place pour les sous-programmes écrits en langage machine (cf dernière partie du chapitre).

- La présence de K indique à l'ordinateur que l'utilisateur veut créer des caractères graphiques (cf chapitre suivant).

Exemples d'utilisation : cf chapitre 3 jeu du 21.

3.3.2 Instructions de contrôle et de branchement

Bien que les instructions de contrôle soient différentes de celles de branchement, il est plus aisé de les étudier conjointement. En effet en BASIC, dans la majeure partie sinon la totalité des cas, une instruction de contrôle se trouve associée à une instruction de branchement.

- Par définition une instruction de contrôle réalise un test ou contrôle.

Exemple : déterminer entre deux joueurs A et B lequel a gagné nécessite une instruction de contrôle.

- Une instruction de branchement s'utilise en général en mode programme. Par définition elle permet de modifier le séquencement de l'exécution d'un programme. En effet, un programme s'exécute ligne par ligne, et normalement dans l'ordre croissant des numéros de ligne (i.e. : ligne 1 puis ligne 2, puis 3, puis 4, etc....). Cette façon de procéder est modifiable par une instruction de branchement. Une telle instruction indique à l'ordinateur que la prochaine ligne à exécuter est celle mentionnée dans cette même instruction (de branchement).

Ces deux types d'instruction, décrites sommairement, sont fondamentaux pour la programmation. Ils permettent de réaliser des programmes capables de décision.

Les instructions de branchement et de contrôle sont les suivantes :

GOTO [numéro de ligne]	Instructions de branchement
GOSUB [numéro de ligne]	

IF [expression logique]	THEN [suite d'instructions] ‹ELSE [suite d'instructions]›	Instruction de contrôle
IF [expression logique]	GOTO [numéro de ligne] ‹ELSE [suite d'instructions]›	
ON [expression logique]	GOTO [suite de numéros de ligne]	Instructions de contrôle et de branchement
	GOSUB [suite de numéros de ligne]	
ON ERROR GOTO	[numéro de ligne]	

Une partie de ces instructions a été abordée dans le chapitre précédent.

* GOTO

Syntaxe GOTO [numéro de ligne]

Elle doit toujours être suivie d'un numéro de ligne, sans quoi le TO7 affiche :

? SN Error (erreur de syntaxe)

Elle donne l'ordre à l'ordinateur de continuer l'exécution du programme au numéro de ligne indiqué. Si ce numéro correspond à une ligne vide, l'ordinateur imprime :

? UL Error (UL pour Undefined Line - ligne vide).

Une utilisation abusive de GOTO est à déconseiller, car elle conduit à des programmes inextricables, comparables à de véritables labyrinthes. Néanmoins le BASIC conduit à une utilisation forcée.

Note : la ligne repérée par le numéro mentionné après GOTO est appelée ligne de branchement.

Exemple :

```
100  IF SCORE A > SCORE B THEN PRINT "A EST
     GAGNANT" ELSE GOTO 200
 :
 :
200  IF SCORE A = SCORE B THEN PRINT "JOUEURS
     EX AEQUO" ELSE PRINT "B EST GAGNANT"
```

La ligne 200 n'est exécutée que si SCORE A est inférieur ou égal à SCORE B.

* GOSUB-RETURN

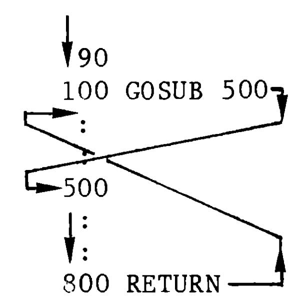
Syntaxe : GOSUB [numéro de ligne]
RETURN

GOSUB est toujours suivi d'un numéro de ligne et la ligne considérée ne doit pas être vide (cf GOTO). Cette instruction produit deux effets sur l'ordinateur :

- A l'exécution il mémorise le numéro de la ligne où cette instruction apparaît.

- Il continue l'exécution à partir du numéro de ligne indiqué et cela jusqu'à ce qu'une instruction RETURN (signifiant Retour) soit rencontrée. Dans ce dernier cas il retourne à la ligne correspondant au numéro (de ligne) mémorisé (cf premier effet). A partir de là il reprend l'exécution normale.

Il est possible de schématiser ceci (sur un exemple) :



Remarque : la partie du programme située entre le numéro de ligne mentionné dans l'instruction GOSUB (500 dans l'exemple) et la première instruction RETURN constitue un sous-programme.

Cette instruction, dont l'utilisation est à recommander, est très importante. Son emploi nécessite le fractionnement de l'organigramme en sous-parties. Une "méthode" de fractionnement fera l'objet de la dernière partie de ce chapitre. Les sous-programmes doivent être placés le plus souvent en fin de programme pour la lisibilité de l'ensemble. En fait un sous-programme est un programme.

Comme il est absurde de mélanger deux programmes, il convient de placer programme et sous-programmes les uns à la suite des autres, pour ne pas les enchevêtrer.

* IF ...THEN ... ELSE ...

Syntaxe :

```
IF              (THEN                    )(ELSE
[expression     ([numéro de ligne]       ) [numéro de ligne])
logique]        ([suite                  ) [suite
                (d'instructions]         )d'instructions])
                (                        )
                ( GOTO                   )
                ([numéro de ligne]       )
```

Une suite d'instructions est une succession d'instructions séparées chacune par deux points.

IF ... THEN ... ELSE est commune à de nombreux langages de programmation tant elle est importante. L'essentiel est de bien comprendre sa signification ; rappelons-la :

<u>si</u> (IF)
l'expression logique est vraie

<u>alors</u> (THEN)
l'ordinateur - exécute les instructions situées après THEN
ou
- saute à la ligne indiquée (cas IF ... GOTO)

<u>sinon</u> (ELSE)
il - exécute les instructions situées après ELSE
- saute à la ligne indiquée

Il faut savoir que toute expression syntaxiquement correcte et logique (utilisant des IF ... THEN ... ELSE ...) est autorisée. Une liste non limitative de cas d'emplois a été donnée au chapitre précédent (cf 2.4.5).

* ONGOSUB-ONGOTO

Syntaxe :

```
ON              (GOSUB [numéro de
[expression     (        ligne] [,[numéro de ligne]]
numérique]      (
                (GOTO  [numéro de
                (        ligne] [,[numéro de ligne]]
```

L'expression numérique est évaluée. Notons V sa valeur. Le terme GOTO (ou GOSUB) est suivi par une liste de numéros de ligne séparés par des virgules ; appelons N le nombre de numéros (de ligne). Plusieurs cas se présentent :

- Si V n'est pas entier l'ordinateur l'arrondira. Dans la suite V est considéré comme entier (s'il ne l'est pas il faut l'arrondir).

- Si V est strictement négatif ou supérieur à 255, le message d'erreur suivant apparaît :

? FC Error (In ...)

OK

- Si V est égal à 0 ou strictement supérieur à N l'exécution saute au premier numéro de ligne indiqué (avec mémorisation du numéro de ligne de départ ; cas ON ... GOSUB).

- Dans les cas restants ($0 \leqslant V \leqslant N$) l'exécution saute à la ligne indiquée par le Vième numéro (avec mémorisation du numéro de ligne de départ pour pouvoir revenir : cas ON GOSUB).

Remarque : il faut que les différents numéros de ligne correspondent à des lignes non vides, sinon après l'essai de saut à cette ligne le TO7 arrête l'exécution et affiche

? UL Error (In ...)

Exemple : le programme suivant détermine si une année est bissextile.

```
10  INPUT "ANNEE" ; AN%
20  PRINT "L'ANNEE" ; AN%
30  ON (AN% MOD 4 + 1) GOSUB 60, 80, 80, 80
40  PRINT "BISSEXTILE"        (si l'année est
                               bissextile, elle
                               est divisible par
                               4)
50  END
60  PRINT "EST ";
70  RETURN
80  PRINT "N'EST PAS ";
90  RETURN
```

* ONERROR GOTO-RESUME

Syntaxe : ON ERROR GOTO [numéro de ligne]
RESUME { [NEXT]
       { [numéro de ligne]

En principe si le TO7 détecte une erreur il affiche un message d'erreur et stoppe toute exécution entreprise. Par l'intermédiaire de cette instruction le

le déroulement du traitement ne sera pas stoppé, mais l'exécution continuera au numéro de ligne indiqué après GOTO, tout en conservant systématiquement en mémoire le numéro de ligne comportant l'erreur. Il faudra de plus inscrire dans le programme, à partir de la ligne de branchement (au moins), une ligne utilisant l'instruction RESUME. En effet il faudra préciser à l'ordinateur ce qu'il doit faire :

- recommencer l'instruction qui a provoqué l'erreur ; pour ce faire il faut écrire :

RESUME

- continuer l'exécution à la ligne suivant "l'instruction erronée" : il faut écrire :

RESUME NEXT

- continuer à un numéro de ligne quelconque : il faut écrire :

RESUME [numéro de ligne]

Remarques :

- Après avoir exécuté l'instruction RESUME, l'ordinateur abandonne l'adresse de la ligne où est apparue l'erreur. Si par suite d'une programmation "défectueuse" le TO7 rencontrait une nouvelle erreur avant RESUME il stopperait l'exécution, et afficherait un message d'erreur. En effet dans ce cas il serait obligé de mémoriser les deux adresses correspondant aux deux erreurs, ce qui est impossible de par sa conception et détermine son arrêt.

- Il est conseillé d'insérer cette instruction (ON ERROR GOTO) en début de programme afin que le TO7 sache dès le commencement de l'exécution qu'il ne doit pas s'arrêter en cas d'erreur.

Exemple :

En reprenant l'exemple précédent (année bissextile) le programme peut être modifié comme suit :

```
 5  ON ERROR GOTO 70
10  INPUT "ANNEE" ; AN%
20  PRINT "L'ANNEE" ; AN%
30  ON (AN% MOD 4)/(AN% MOD 4)
    GOSUB 80            (si l'année est bissex-
                        tile AN% MOD 4 = 0, la
                        décision en ligne est
                        impossible)
40  PRINT "BISSEXTILE"
50  END
60  PRINT "EST_" ;
70  RESUME NEXT (ou RESUME 40)
80  PRINT "N'EST PAS_" ;
90  RETURN
```

* ERR-ERL

Il s'agit des deux variables dans lesquelles le TO7 place le nom (ERR) (et l'adresse (ERL) en mode programme uniquement) de la dernière erreur rencontrée.

Remarque : l'instruction ERR peut être utilisée conjointement avec une instruction ON ERROR GOTO En effet elle permet d'effectuer des branchements différents suivant le type d'erreur rencontré.

Exemple(*) :

```
10      ON ERROR GOTO 300
:
:
300  IF ERR = 3 THEN PRINT "IL MANQUE UN
     END DANS LE PROGRAMME"
310  IF ERR = 57 THEN PRINT "LE FICHIER
     N'EST PAS OUVERT"
320  RESUME
```

* ERROR

Syntaxe : ERROR [expression numérique]

Elle permet de simuler l'erreur indiquée par la valeur de l'expression numérique.

Remarque : il peut y avoir une erreur dans l'instruction elle-même.

En effet la valeur de l'expression numérique doit être comprise après arrondi éventuel entre 1 et 23 inclus ou entre 50 et 61 inclus. Si la valeur n'est pas correcte l'erreur ? UE ERROR est affichée.

Exemple*:

```
10 ERROR 2
```

Ce "programme" affichera

```
? SN Error
```

(*) Pour les codes d'erreur se reporter à l'annexe C.

3.3.3 Les boucles FOR... NEXT

Une boucle est une partie du programme qui pourra être répétée au cours de l'exécution. Une telle partie est délimitée par deux lignes comportant des termes particuliers, à savoir :

. FOR - TO - STEP -(à son début)

. NEXT - (à sa fin).

Pour compléter ces "termes" il est nécessaire d'introduire des variables entières ou réelles (qui sont les seules autorisées, sinon le TO7 affiche : ? TM Error)

```
FOR ... TO ... (STEP ...)
 :
NEXT
```

Syntaxe :

```
FOR [nom de variable numérique] = X TO Y
⟨STEP Z⟩
 :
 :
NEXT ⟨[nom de variable ⟩⟨,[nom de variable ⟩
     ⟨       numérique]⟩⟨          numérique]⟩
```

(X, Y et Z peuvent être des constantes ou des expressions numériques).

Remarque préliminaire : si l'instruction STEP 2 est omise le TO7 considérera d'autorité que Z vaut 1.

Cette instruction signifie :

- faire varier la variable numérique de X à Y par incréments de Z, et cependant
- exécuter les instructions inscrites dans la boucle.

Pour que cette boucle soit exécutée au moins une fois il n'y a que deux cas possibles :

1. X est inférieur ou égal à Y ($X \leqslant Y$) et le pas (Z) est positif.
   (note : si X est égal à Y la boucle ne sera exécutée qu'une fois).

2. X est supérieur ou égal à Y ($X \geqslant Y$) et le pas (Z) est négatif.
   (même note que ci-dessus).

On peut souvent (ex. : au court de manipulations de matrices ayant plus de deux dimensions) avoir besoin de plusieurs boucles en même temps. Dans ce cas, il ne faut pas oublier la règle suivante :

Des boucles ne doivent jamais s'entrelacer, mais elles peuvent être emboîtées.

Exemple 1 :

```
10   DIM A(3,3)

20   FOR I = 1 TO 3                        ) Cette boucle
                                           ) fait varier
30   FOR J = 1 TO 3 ) Cette boucle         ) le numéro de
                    ) remplit les          ) ligne de 1 à
40   A (I, J) = 1   ) cases de la          ) 3 ;
                    ) ligne I avec         ) cependant la
50*  NEXT J         ) des 1                ) ligne est
                                           ) remplie avec
60   NEXT I                                ) des 1
```

* Les lignes 50 et 60 peuvent être remplacées par :

```
50   next J, I
```

Exemple 2 :

```
10   FOR I = 1 TO 3 STEP 0.1

20   FOR J = 1 TO 4

30    :

40    :

50   NEXT I

60   NEXT J
```

Dans ce deuxième exemple, les boucles sont enchevêtrées, et de ce fait le TO7 affichera :

NF Error In 50 (NF pour Next without For)

L'ordinateur n'a pas trouvé le Next attendu (à savoir Next J).

Remarque : il doit toujours y avoir un NEXT à la suite d'un FOR sinon le message d'erreur : ? FN Error sera affiché (FN pour For without Next).

3.3.4 Instructions d'entrée-sortie

Elles permettent d'établir le dialogue entre l'utilisateur et l'ordinateur. Il faut distinguer les instructions d'entrée de celles de sortie.

— INSTRUCTIONS D'ENTRÉE

Elles servent à introduire dans l'ordinateur (l'unité centrale) les données (au sens large du terme).

* INPUT

Syntaxe :

INPUT (;) [" chaîne de caractères "] {,;} variable, [variable] ...

Remarques : le premier point-virgule est sans objet avec le BASIC du TO7. Comme il est nécessaire pour d'autres BASIC il a été conservé en option pour respecter la compatibilité des programmes.

Lors de l'exécution de l'instruction l'ordinateur imprime la chaîne de caractères (suivie d'un point d'interrogation si on a placé un point-virgule à la suite de cette chaîne dans le programme). Puis il attend que l'utilisateur frappe sur la touche [ENTREE] pour pouvoir reprendre l'exécution. Deux cas se présentent :

- l'utilisateur a tapé correctement le jeu de données qu'attendait l'ordinateur et l'exécution reprend.

- l'utilisateur n'a pas tapé correctement le jeu de données attendu et le message suivant est affiché:

? Redo
[chaîne de caractères] ?

Le TO7 considère la première (fausse) entrée comme nulle et redemande le jeu de données jusqu'à obtenir une instruction correcte. Une mauvaise introduction peut se produire de deux manières :

- l'utilisateur a introduit trop ou pas assez de données.

- l'utilisateur a introduit des données de types différents de celles attendues et l'ordinateur n'a pas pu réaliser la conversion de type comme lorsqu'il reçoit une chaîne de caractères au lieu d'une valeur numérique (attendue).

Remarques :

- L'introduction des données peut être interrompue en frappant simultanément sur les touches [CNT] et [C]. En cas de reprise de l'exécution par l'instruction CONT il faudra réintroduire en entier les jeux de données.

- Un tableau s'introduit variable par variable.

- Une chaîne de caractères doit en principe être encadrée par des guillemets. Ces derniers ne sont pas utiles si la chaîne ne contient pas de caractères blancs (espacements) à son début et à sa fin.

Exemple 1 :

Considérons le début de programme suivant :

```
10 INPUT "MISE, NOM" ; MISE%, NOM$
20 ...
```

Si à l'exécution (après affichage de la question MISE, NOM ?) l'utilisateur tape 50, JEAN [ENTREE] l'ordinateur place dans la mémoire MISE% la valeur 50 et la chaîne "JEAN" dans NOM$, et il n'y a pas d'erreur d'introduction.

Exemple 2 : Si avec le même début de programme que ci-dessus l'utilisateur tape

50, JEAN, 3 [ENTREE]

l'ordinateur n'acceptera pas cette introduction car il y a trois données alors que deux seulement sont attendues.

* INPUTWAIT

Syntaxe :

INPUTWAIT [numéro de ligne] ; [durée] [, [variable] ...]

Elle a la même fonction que l'instruction INPUT mais elle permet de limiter le temps alloué à l'utilisateur pour introduire les données. Par exemple, elle peut être mise à profit pour des jeux basés sur les réflexes, l'utilisateur ayant à répondre en un temps aussi court que possible.

Dans la syntaxe (cf ci-dessus) [durée] est soit une constante, soit une variable, soit une expression numérique ; elle représente le temps maximum, exprimé en secondes, dont dispose l'utilisateur pour entrer ses données. Si ce dernier le dépasse, l'exécution du programme reprend à la ligne indiquée par [numéro de ligne], sinon l'ordinateur passe à l'instruction suivante (si toutefois les données sont correctement introduites, sans quoi il répète l'instruction).

Exemple :

Ce programme teste votre rapidité.

```
10  PRINT "VOUS DEVEZ TAPER :"BONJOUR" AUSSI
    VITE QUE POSSIBLE DES QUE LE POINT D'IN-
    TERROGATION SERA AFFICHE. VOUS AVEZ DROIT
    A TROIS ERREURS" :
20  NBC = 0 : TEMPS = 10
30  INPUTWAIT 70 ; TEMPS, REP$
40  IF REP$<>"BONJOUR" THEN IF NBC = 4 THEN
    GOTO 70 ELSE NBC = NBC + 1 : GOTO 30
50  TEMPS = TEMPS - 1
60  GOTO 30
70  PRINT "VOTRE TEMPS MINIMAL EST" ; TEMPS
80  END
```

Avec le programme précédent l'ordinateur vous demande d'imprimer BONJOUR en un temps inférieur à TEMPS. Au départ TEMPS vaut 10 puis tant que vos réponses sont justes (tant que vous tapez BONJOUR dans le temps imparti et sans faute) il diminue le temps d'une seconde (chaque fois). Lorsque vous n'êtes plus assez rapide ou lorsque vous avez commis trois fautes (en tapant une autre chaîne de caractères que celle demandée : BONJOUR) l'ordinateur imprime votre temps minimal puis s'arrête.

Remarque : si l'indication [numéro de ligne] est omise, le TO7 affiche lors de l'exécution

? UL Error In X

(X est le numéro de la ligne comportant l'instruction incorrecte).

* LINE INPUT (ou LINEINPUT)

Syntaxe :

```
{[LINE INPUT]}  [;][chaîne de caractères] {,}
{[LINEINPUT] }                            {;}
                            [variable chaîne]
```

Remarques :

- Cf remarque 1 instruction INPUT.

- Contrairement à l'instruction INPUT aucun point d'interrogation n'apparaît à l'exécution (seule la chaîne de caractères est affichée).
- La virgule et le point-virgule sont interprétés de la même manière par le TO7.

L'instruction LINE INPUT permet d'introduire une chaîne d'au plus 255 caractères. Cette chaîne n'a pas à être encadrée par des guillemets (contrairement à INPUT), et de plus elle peut comporter tous les caractères sans restriction. C'est donc une instruction adaptée à l'introduction de chaînes de caractères.

Exemple :

```
10 LINEINPUT A$
20 PRINT A$
```

A l'exécution si vous tapez

2 * 3 + 4 [ENTREE]

vous obtiendrez à l'affichage :

2 * 3 + 4

OK

* READ-DATA-RESTORE

Syntaxe :

READ [variable] [,[variable]] ...

Cette instruction est à utiliser avec deux autres, à savoir : DATA et RESTORE (DATA signifie : données).

Syntaxes :

DATA [constante] (,[constante]) ...

RESTORE ‹[numéro de ligne]›

Parfois dans certains programmes il existe des jeux de données qui ne varient pratiquement jamais. Ils peuvent être introduits directement dans l'écriture du programme ; cela permet de ne pas avoir à les réécrire (réintroduire) à chaque exécution : c'est l'objet des trois instructions présentées ci-dessus.

L'instruction DATA permet d'inscrire les données dans le programme. Il faudra créer une ou plusieurs ligne(s) de programme dans laquelle (lesquelles) la (les) donnée(s) seront inscrite(s) à la suite de DATA.

Exemple :

```
100 DATA 3.14159 , 365, "MOT MAGIQUE"
110 DATA "PI", 2.71828
```

L'instruction READ (qui signifie lire en anglais) permet de lire les données. A chaque instruction READ le TO7 lira dans les lignes repérées par DATA les premières valeurs qui n'auront pas déjà été lues (par une instruction READ antérieure). Il faut que le type de la variable (dans laquelle va être mis l'élément lu) et le type de la donnée soient compatibles.

Exemple :

```
 20 READ A, B$
 30 READ NOTE%
  :
100 DATA 100, "JO"
110 DATA "DAPHNIS", "ZERO", 32767
```

Le programme contient une erreur : il est demandé en ligne 20 et 30 de lire successivement un réel, une chaîne de caractères et un entier ; or les instructions DATA contiennent successivement un réel, trois chaînes de caractères et un entier. Au moment de la lecture de "l'entier" (NOTE%) le TO7 va trouver une chaîne de caractères, d'où le message d'erreur.

```
? SN Error In 110

OK
```

Remarque : si le nombre d'éléments à lire est supérieur au nombre de données, l'ordinateur affiche

? OD Error (OD pour Out of Data)

OK

Cependant le TO7 peut recevoir l'ordre de relire plusieurs fois un même jeu de données : c'est le but de l'instruction RESTORE.

Après RESTORE la lecture reprendra :

- au début de la première instruction DATA si aucun numéro de ligne n'est précisé (cf syntaxe).

- au début de la première instruction DATA située après le numéro de ligne si ce dernier est précisé (si ce numéro de ligne ne correspond pas à une instruction DATA, il faut qu'il corresponde à une ligne existant dans le programme, sinon le TO7 affiche :

  ? UL Error (UL pour Undefined Line ou ligne inexistante).

Exemple :

```
 10  READ A, B$, NOM$, PI
  :
 50  RESTORE 120 (RESTORE 130 serait plus
 60  READ PI        correct au point de vue
  :                 style de programmation)
  :
110  DATA 100, "JO", "DAPHNIS"
120  PRINT "BONJOUR"
130  DATA 3.141592653589793
```

Après l'exécution de la ligne 50 le TO7 lira les données situées en ligne 130 (premier DATA situé après la ligne 120).

Remarques :

- Pour changer les jeux de données, il faut les modifier dans le programme.

- Il est conseillé de rassembler les instructions DATA et de les mettre à la fin du programme.

* INPUT$

Cette fonction permet de lire une chaîne de caractères, et les caractères tapés ne sont pas affichés à l'écran. Cette dernière remarque induit l'utilisation de mots de passe, de codes ("secrets"), etc...

Exemple d'application : jeu du pendu, mastermind dans lesquels un joueur doit découvrir une combinaison secrète (qui doit demeurer masquée) :

```
10  CODE$ =  INPUT$ (10)
20  PRINT "LE CODE EST INTRODUIT : A VOUS DE
    JOUER"
30   :
```

La ligne 10 permet de mettre dans CODE$ une chaîne de 10 caractères tapée au clavier (celle-ci ne sera pas affichée à l'écran).

Remarques :

Dans ce dernier exemple, l'utilisateur pourra constater, en faisant tourner le programme, qu'à l'exécution de la ligne 10 l'ordinateur attend les 10 caractères ; dès qu'ils ont été tapés au clavier, le TO7 reprend l'exécution sans attendre que l'utilisateur tape sur la touche [ENTREE].

C'est le seul cas où la pression de la touche de validation [ENTREE] n'est pas utile.

Contrairement au cas des deux instructions précédentes, pour INPUT$ le nom de la variable dans laquelle va figurer la chaîne de caractères est situé avant le nom de l'instruction, et le signe = est inséré (l'instruction d'assignation LET est sous-entendue , ci-après).

- INSTRUCTIONS DE SORTIE

Elles sont destinées à extraire de l'unité centrale les résultats obtenus par le calculateur.

* PRINT-TAB

Syntaxe :

$$\text{PRINT}\left[\begin{Bmatrix}\text{[TAB (variable numérique)]}\\ \text{[expression]}\end{Bmatrix}\right]\begin{Bmatrix},\\ ;\end{Bmatrix}\cdots$$

Elle permet d'afficher sur l'écran de contrôle des résultats ou des suites de caractères (dont la taille est à préciser grâce à l'instruction ATTRB, cf ci-après). Le terme <u>expression</u> situé dans la syntaxe est à prendre au sens large ; il signifie en même temps : expression logique, expression chaîne de caractères, expression numérique.

Exemples d'expressions :

1.
```
10 INPUT "MISE" ; MISE
20 PRINT MISE>= 0
```

Ce petit programme affiche - 1,ou 0 si la mise est négative.

2.
```
10 INPUT "CARACTERES" ; A$
20 PRINT A$ + A$
```

3. En tapant

PRINT (2 * 3 + 5 * 3)/7 [ENTREE] on obtient

3

OK

Remarque : les expressions peuvent contenir des fonctions telles que cosinus, sinus, etc... (cf ci-après en ce qui concerne les fonctions).

La fonction TAB à laquelle il est fait référence (dans la syntaxe) permet d'écrire exactement à l'endroit désiré, sans restriction. Les lignes sur l'écran de télévision comportent 40 positions, elles sont notées de 0 à 39. L'insertion de l'instruction TAB (I) où I est une grandeur numérique comprise entre 0 et 39, permettra de positionner le curseur en place I.

Remarques sur les valeurs de I ; si :

- elle n'est pas entière, le TO7 prendra l'entier le plus proche ;
- elle est strictement négative, il la considérera comme nulle ;
- elle dépasse 39, la valeur prise sera le reste de la division entière de I par 40.

Exemple :

PRINT TAB (25) "BONJOUR" [ENTREE]

permettra d'imprimer le mot BONJOUR à partir de la position 25.

En utilisant l'instruction PRINT seule (sans TAB), trois formes d'impression sont possibles, à savoir :

- Imprimer tout à la suite sans espaces. Cette impression s'obtient en séparant les différents éléments à imprimer par des points-virgules.

Exemple :

```
PRINT "BONJOUR" ; "MONSIEUR" [ENTREE]
```

donne

```
BONJOURMONSIEUR

OK
```

Remarque :

Dans un programme, si le dernier élément à imprimer est suivi d'un point-virgule, la prochaine impression se fera à la suite, sans espace, sinon il y aura passage à la ligne.

Exemples :

1.-
```
10  PRINT "BONJOUR_" ;
20  PRINT "COMMENT ALLEZ-VOUS ?"
30  PRINT "MONSIEUR"
```

A l'exécution le programme donne :

```
BONJOUR COMMENT ALLEZ-VOUS ?
MONSIEUR

OK
```

2.- 10 PRINT "BONJOUR" ;
    20 PRINT "MONSIEUR"

L'exécution donne

BONJOURMONSIEUR

OK

- Imprimer avec des espaces entre chaque élément. Il faut utiliser des virgules à la place de points-virgules. Le TO7 fractionne en trois zones les 40 positions d'affichage.

la première va de 0 à 13 (inclus)
la seconde va de 13 à 25 (inclus)
la troisième va de 26 à 38 (inclus)

Avec la présence d'une virgule, lorsque l'impression d'un élément est terminée, il passe au début de la zone suivante pour en imprimer d'autres.

Remarque :

Si dans un programme le dernier élément à imprimer (d'une instruction PRINT) est suivi d'une virgule, l'affichage suivant se fera dans la zone suivante sinon il y a passage à la ligne.

Exemple

```
10  PRINT "AU REVOIR", "MONSIEUR_";
20  PRINT "A",
30  PRINT "BIENTOT"
```

Ce programme donne à l'exécution :

```
AU REVOIR     MONSIEUR A    BIENTOT
^             ^             ^
0             13            26
```

Remarques :

- Le TO7 "passe" à la ligne automatiquement lorsqu'il n'y a plus assez de place sur une ligne pour afficher ce qui est demandé.

- Les valeurs numériques ne peuvent être affichées sur plus d'une ligne ; elles sont précédées d'un

signe - si elles sont négatives
blanc si elles sont positives

- Imprimer une ligne sans caractère (i.e. faire un saut de ligne). Il suffit de placer (d'introduire) l'instruction PRINT "seule".

* PRINTUSING

Syntaxe:

PRINTUSING [expression chaîne];[X]⟨;⟨[Y]⟩⟩...

X et Y sont des expressions au sens "large" cf PRINT.

Elle offre beaucoup de possibilités d'affichage. Il faut introduire dans l'instruction la description de la façon de disposer l'affichage. Cette description est faite au moyen d'une image.

Par exemple le TO7 peut afficher les résultats avec un certain nombre de chiffres après la virgule, ou encore utiliser la notation dite scientifique etc...

Une image est constituée par une chaîne de caractères. Il peut y avoir plusieurs images dans ce cas :

elles seront placées les unes à la suite des autres sans aucun caractère séparateur (entre elles). L'image ou l'ensemble des images (suivant le cas) sera placé entre guillemets. Il existe deux types d'images, respectivement destinées à l'affichage :

- des valeurs numériques
- des chaînes de caractère

Elles sont constituées par des caractères à choisir parmi :

. + - $$ **, ↑↑↑↑ pour les premières ;

! % & _ (ce dernier représente un caractère blanc) pour les secondes.

Les différentes images possibles étant décrites en détail dans le manuel de référence, nous ne les reprendrons pas ici.

Exemples :

1. PRINTUSING " #.#" ; 3.141 [ENTREE]

donne 3.1 (l'image est #.# et elle indique qu'il ne faut qu'un chiffre après le
OK point-virgule)

2. PRINTUSING "!" ; "BEAU" [ENTREE]

donne B (l'image est !, elle indique qu'il ne faut conserver que la première lettre
OK de la chaîne donnée)

Il faut prendre garde à ce que le type de l'image donnée corresponde au type d'élément à afficher (soit valeur numérique, soit chaîne de caractères). Les images données sont utilisées les unes à la suite des

autres. Si elles ont été toutes utilisées et qu'il reste encore des informations à afficher, le TO7 reprend le début de la suite des images jusqu'à épuisement des éléments à imprimer.

Exemple :

PRINTUSING "# . # !" ; 3 * 1.41; "BEAU"; 2.718; "JO" [ENTREE] donne

4.2 B 2.7 J

OK

Il y a deux images (à savoir # . # et !) et quatre éléments à afficher, le TO7 se sert deux fois de chaque image ; après l'affichage de B il reprendra la première image.

Remarques :

- Si aucune image n'est fournie l'erreur de syntaxe SN est détectée.

- Si le type d'image ne correspond pas au type de l'élément à afficher le message d'erreur apparaît

? TM Error

OK

* ATTRB UNMASK

Syntaxes ATTRB [X], [Y], [Z] UNMASK

Elle permet de définir la taille des caractères affichés sur l'écran de télévision, ainsi que le mode d'affichage (masqué ou non masqué). Les valeurs possibles pour X, Y, et Z sont 0 et 1.

X définit la largeur des caractères
(0 largeur normale ; 1 largeur double)

Y définit la hauteur des caractères
(0 : hauteur normale ; 1 hauteur double)

Z définit le mode d'impression
(0 : mode normal ; 1 mode masqué)

Remarques :

- Lors de l'utilisation de caractères de largeur double une ligne ne contient que 20 (soit 40/2) caractères au maximum (cf instruction PRINT).

- Pour quitter le mode masqué il suffit d'introduire l'instruction UNMASK ou SCREEN.

Exemple :

```
10 ATTRB , 1, 1
```

Ordonne à l'ordinateur d'imprimer les caractères avec une hauteur double et en mode masqué.

* CLS

Syntaxe CLS

Elle nettoie la fenêtre de travail (à l'écran) et elle positionne le curseur en haut et à gauche de cette fenêtre.

(Exemple : cf chapitre 2, jeu du 21)

3.3.6 Commandes de mise au point d'un programme

Dans la suite de ce paragraphe les termes fichier et descripteur de fichier vont être cités assez souvent. Ces termes seront décrits en détail dans le paragraphe "Entrée-Sorties Généralisées". Pour l'instant on retiendra simplement les définitions suivantes :

- Fichier, ou plus exactement nom de fichier, représente le nom sous lequel est mémorisé (sur cassette, disquette, ...) ou transféré (vers la voie série, imprimante, écran, ...) un programme ou des données.

Un nom de fichier est constitué par une suite de six caractères suivis d'un point et d'un suffixe optionnels indiquant s'il s'agit d'un fichier BASIC (BAS), de Données (DAT) ou Binaire (BIN). (Binaire se rapporte aux programmes écrits en langage machine).

Exemple : Un programme de jeu peut être mémorisé sur cassette sous le nom de fichier "JEUX.BAS".

- Descripteur de fichier

Il contient le nom de fichier et le nom du type de périphérique (cassette (CASS), imprimante (LPRT), voie série (COMM), écran (SCRN), etc...) dans lequel est situé ou transféré le programme ou les données. L'ensemble de ces deux noms est placé entre des guillemets.

Exemple : "CASS : JEUX.BAS"

périphérique fichier

* LIST

LIST a deux utilisations possibles.

. LIST ⟨[numéro de ligne]⟩⟨-[numéro de ligne]⟩

Celle-ci permet de lister sur l'écran tout ou partie du programme qui se trouve en mémoire centrale.

Exemples :

LIST [ENTREE] imprime tout le programme
LIST 40-50 liste les lignes 40 à 50
LIST -50 liste de la ligne 1 à la ligne 50

. LIST ⟨ [descripteur de fichier]⟩⟨[plage de numéros de ligne]⟩

Elle permet d'imprimer sur le périphérique concerné tout ou partie du programme situé en mémoire centrale : celui-ci sera placé dans le fichier indiqué (dans le descripteur).

Remarques :

1. Cette commande est utilisable avec : l'écran, le lecteur-enregistreur de cassettes, l'imprimante parallèle, et la voie série.

2. Si aucun périphérique n'est mentionné l'ordinateur considère d'autorité l'écran.

3. Les lignes à imprimer sont déterminées par [plage de numéro de ligne].

Exemple :

LIST "CASS : ESSAI.ONE", 50-100

permet d'imprimer sur cassette les lignes 50 à 100 du programme situé en mémoire centrale. Le nom de fichier sera ESSAI.ONE.

* DELETE

Syntaxe :

DELETE ⟨[numéro de ligne]⟩⟨-⟨[numéro de ligne]⟩⟩

Elle est utilisée en mode calcul uniquement, et elle permet de détruire les lignes de programme mentionnées dans l'instruction.

Exemples :

DELETE [ENTREE] détruit tout le programme
DELETE 40-50 supprime les lignes 40 à 50
DELETE 45- supprime toutes les lignes à partir de la ligne 45.

* END

C'est l'instruction qui arrête l'exécution d'un programme.

Remarques :

- Cette instruction ferme les fichiers ouverts en cours de programme.

- Si elle n'est pas toujours nécessaire, il vaut mieux prendre l'habitude de l'utiliser, du moins pour les débuts en programmation.

Exemples :

```
1.        10  PRINT "BONJOUR"
          20  END  (ici END n'est pas utile)

2.       100  GOSUB 1000
           :
        1000
           :
        1100 RETURN
```

Si aucun END n'est placé dans le programme (entre les lignes 0 et 999), l'ordinateur considérera, lors de l'exécution, les lignes 1000 à 1100 comme étant du programme, ce qui conduit le TO7 à afficher :

? RG Error In 1100   (il y a un return sans GOSUB)

* STOP

Cette instruction peut être utilisée au clavier (touche [STOP]) ou dans un programme (instruction STOP). Elle arrête l'exécution et le message suivant apparaît :

Break In [numéro de ligne]

* CONT

(Mis pour CONTinue) (Syntaxe CONT)

Cette instruction :

- permet de reprendre l'exécution d'un programme s'il a été stoppé ; dans ce cas elle reprend à partir de l'endroit où le TO7 s'est arrêté.

n'est pas utilisable en mode programme.

Remarque : si le numéro de ligne où s'est produit l'arrêt est connu (appelons-le n), alors CONT est équivalente à :
GOTO n

* REM

Syntaxe {REM / ,} [texte]

Cette instruction permet d'introduire du texte dans un programme (le texte est écrit à la suite de REM). Il faut autant de REM qu'il y a de lignes de commentaires.

Exemple : cf chapitre 2, jeu du 21.

Remarque : REM peut être remplacé par une virgule.

* TRON-TROFF

Elles permettent de passer en mode Trace, et de le quitter. Le mode Trace se caractérise par le fait que l'ordinateur affiche à l'écran chaque numéro de ligne exécutée. Ce mode est très utile pour la recherche d'erreurs dans un programme. En effet il permet de vérifier que l'ordre d'exécution des instructions est en accord avec celui que désire le programmeur.

Les instructions suivantes sont relatives à la numérotation automatique, à la sauvegarde de programmes, au listage de programmes, etc...

* AUTO

Syntaxe AUTO [I] [[, J]]

Elle permet de passer en mode programme et de numéroter AUTOmatiquement les lignes (de programme). Les valeurs de I et J sont entières et strictement positives ; si l'une d'elles est omise, le TO7 la considérera comme égale à 10. La numérotation commencera à la ligne numéro I, et l'incrément des numéros sera J.

Exemples :

1. AUTO 5,5

donnera lieu à une numérotation des lignes de programme de 5 en 5 à partir de ligne 5 (soit 5,10,15, etc...).

2. AUTO 5 est équivalent à AUTO 5, 10

Remarque :

Il est recommandé d'utiliser AUTO 10,10 ou simplement AUTO (pour laisser ainsi une place suffisante entre deux lignes de programme).

* NEW

Syntaxe : NEW

Elle vide de son contenu la partie de la mémoire réservée à l'utilisateur. Ainsi le programme et toutes les variables sont détruits.

Remarque : les commandes faites par une instruction CLEAR antécédente ne sont pas oubliées par le TO7. En effet, celles-ci ne sont pas mémorisées dans la zone mémoire réservée à l'utilisateur.

* MERGE

Syntaxe : MERGE ‹[descripteur de fichier]‹,R››

Dans certains cas il est intéressant de fusionner deux programmes (ex. : un programme est enregistré sur cassette, un autre vient d'être écrit et il utilise le premier comme sous-programme ; dans ce cas il est inutile de taper le premier programme une deuxième fois). En effet avec la commande MERGE le TO7 cherche sur le périphérique indiqué le fichier désiré (ces deux derniers étant ceux mentionnés dans le descripteur de fichier). Lorsqu'il l'a trouvé, il le place en mémoire programme tout en conservant celui qui s'y trouvait déjà, puis lance l'exécution si ,R est ajouté.

Remarques :

1. Les seuls périphériques utilisables avec cette instruction sont . le lecteur-enregistreur de cassettes,
. la voie série.

2. Si aucun fichier n'est mentionné dans le descripteur, le TO7 prend le premier (fichier) trouvé (sur le périphérique).

Si aucun périphérique n'est indiqué (dans le descripteur de fichier) l'ordinateur considérera d'autorité le lecteur-enregistreur.

3. Seuls les programmes conservés dans les périphériques sous forme ASCII (cf instruction SAVE) sont fusionnables ; pour les autres l'erreur ? FM Error est détectée.

4. Si les deux programmes contiennent des lignes de programme ayant les mêmes numéros, le TO7 ne conservera que celles issues du programme stocké sur le périphérique (celles du programme principal sont éliminées).

* RUN

RUN a deux utilisations possibles :

1. RUN ⟨[numéro de ligne]⟩ lance l'exécution du programme, chargé en mémoire centrale, à partir de la ligne indiquée par [numéro de ligne] si elle est précisée, sinon à partir de la ligne 1.

2. RUN [descripteur de fichier] ⟨,R⟩ cherche dans le périphérique mentionné le fichier désiré (ces deux derniers sont à préciser dans le descripteur de fichier). Après avoir trouvé il lance l'exécution.

Remarques :

- cf remarque 1 commande MERGE (ci-avant)
- cf remarque 2 " " "
- la commande RUN ferme tous les fichiers qui étaient déjà ouverts, sauf si ,R est ajouté.

Exemple :

```
RUN "CASS : (5880) ESSAI.ONE"
```

* SAVE

Syntaxe :

SAVE [descripteur de fichier] { ,<A> / ,<P> }

Cette instruction permet de sauvegarder (i.e. : enregistrer) le programme qui se trouve en mémoire centrale sur le périphérique mentionné et sous le nom de fichier indiqué.

Remarques :

1. Les périphériques utilisables avec cette instruction sont : l'écran, le lecteur-enregistreur, l'imprimante parallèle et la voie série.

2. Cf remarque 2 commande MERGE.

3. Si le terme optionnel A (resp P) est indiqué, le TO7 conservera le programme sous forme ASCII (resp sous forme protégée). Dans ce dernier cas,

- le listage
- la modification de programme
- l'utilisation des fonctions PEEK et POKE (sur les mémoires concernant le programme)

seront impossibles.

4. Le nom de fichier doit toujours être mentionné dans le descripteur de fichier, sans quoi le TO7 affiche

? FD Error

Exemple :

SAVE "COMM : (5880) JEU" ,A

conserve un programme sous forme ASCII sur la voie série, le nom étant JEU.

* SAVEM

Syntaxe :

SAVEM "[nom de fichier]",[adresse 1], [adresse 2],[adresse d'exécution]

Elle permet de sauvegarder un programme binaire (sur le lecteur-enregistreur exclusivement) sous le nom "[nom de fichier]". Ce programme est placé dans les mémoires numéro [adresse 1], [adresse 1] + 1 ... jusqu'à [adresse 2]. L'adresse d'exécution du programme étant [adresse exécution].

Exemple :

```
10 SAVEM "ESSAI" &H6200, &H75A0, &H0000
```

permet de sauvegarder le programme inscrit dans les mémoires 6200, 6201, ..., 75A0 du fichier ESSAI, l'adresse d'exécution étant 0000.

* LOAD

Syntaxe : LOAD ⟨[descripteur de fichier]⟨,R⟩⟩

Elle permet de chercher, dans le périphérique mentionné, le fichier indiqué (dans le descripteur de fichier). Les fichiers, rencontrés par le TO7, qui ne correspondent pas à celui désiré, sont mentionnés sur

l'écran par le message Skip [nom de fichier]. Tant que le fichier n'est pas trouvé l'ordinateur affiche également le terme : Searching. Dès qu'il l'a trouvé, il écrit : FOUND [nom de fichier]. Puis il le charge en mémoire programme, après quoi il indique OK.

Exemple :

LOAD "COMM : (5880) ESSAI.ONE"[ENTREE]

charge le fichier ESSAI ONE situé sur la voie série.

Remarques

1. Cf remarque 1 commande MERGE

2. " " 2 " "

3. Si le fichier n'est pas trouvé après examen de tout ce qui se trouvait dans le périphérique, il faut réinitialiser le TO7.

4. Cf remarque 3 commande RUN.

* LOADM

Syntaxe :

LOADM ("[nom du fichier]")((,[décalage])(,R))

Elle permet de charger un programme binaire dans la mémoire centrale. Le terme [décalage] indique de combien de cases mémoires le chargement doit être décalé en mémoire. Elle n'est utilisable qu'avec le lecteur-enregistreur.

Exemple :

LOADM "ESSAI", &H0100

charge en mémoire centrale le fichier ESSAI et déplace son chargement de 100 cases mémoires (cf SAVEM).

Remarque : Cf remarque 3 de RUN

* MOTORON-MOTOROFF

Elles permettent respectivement de faire défiler et d'arrêter le défilement du lecteur-enregistreur. Ce dernier doit être connecté et les touches "lecture" ou "lecture et enregistrement" doivent être enfoncées.

3.4 NOTIONS DE SOUS-PROGRAMME ET FONCTIONS

Malheureusement il est difficile, sinon impossible, de donner une méthode systématique de création de sous-programmes. La meilleure façon de l'apprendre est de s'y entraîner. Cependant des critères généraux permettent de savoir si la solution obtenue est acceptable. Une énumération non limitative des règles est proposée ci-dessous :

- Le fractionnement du programme en sous-programmes doit être effectué lors du passage de l'organigramme au programme.

- Les tailles des différents sous-programmes ne doivent pas être trop disparates. Sans quoi il faut vérifier :

  . si les petits sous-programmes ne réalisent pas des fonctions ridiculement simples.

  . si les gros ne peuvent pas être à leur tour fractionnés en plusieurs parties.

- Un sous-programme doit réaliser une seule fonction à la fois.

- La fonction réalisée doit être indépendante de celles du reste du programme.

- Cette fonction doit être relativement simple.

- Il faut éviter qu'un sous-programme utilise des sous-sous-sous programmes etc...

Parmi les sous-programmes qu'il est possible d'insérer dans un programme, l'utilisateur peut en utiliser de trois types :

- Les sous-programmes écrits à la suite du programme principal (cf instructions GOSUB RETURN).

- Les sous-programmes écrits en langage machine.

- Les sous-programmes préenregistrés dans le TO7.

3.4.1 Sous-programme en langage machine

Ceux-ci se définissent au moyen de la fonction DEF USR ; ils s'utilisent grâce à USR ou EXEC.

* **DEF USR** ([numéro]) = (adresse mémoire)

Elle permet de définir jusqu'à 10 sous-programmes qui seront désignés par un numéro (variant de 0 à 9). L'adresse mémoire est l'adresse à partir de laquelle le programme est situé.

Remarque : si le numéro est omis, l'ordinateur considèrera d'autorité 0.

* **USR**([numéro])([argument])

Cette fonction permet de faire exécuter le sous-programme indiqué par [numéro]. Le terme [argument] désigne la variable qui va être traitée au cours du sous-programme assembleur, c'est-à-dire écrit en langage machine.

* **EXEC** [adresse mémoire]

Cette instruction permet le branchement vers un sous-programme. L'adresse mémoire représente l'adresse du début de ce sous-programme.

Exemple :

EXEC 28000

3.4.2 Fonctions

Celles-ci sont appelées des sous-programmes pré-enregistrés dans le TO7. Une liste de ces fonctions est fournie ci-après.

* ABS(X)

  - calcule la valeur absolue de X

* ASC(X$)

  - donne le code ASCII du premier caractère de la chaîne X$

* CDBL(X)

  - convertit le nombre X en autant de types double précision

* CHR$(X)

  - convertit le code ASCII en le caractère associé (pour ceci il faut se référer au codage ASCII)

* CNIT(X)

  - arrondit le nombre X

* COS(X)

  - cosinus (X)

* CSNG(X)

  - transforme le nombre X en un réel simple précision

* CSRLIN

  - donne le numéro de ligne sur lequel se trouve le curseur

* EOF (numéro de canal)

  - permet de savoir si le fichier qui est connecté par le canal (indiqué par numéro de canal) est vide ou non (cf § 3.5)

* EXP(X)

  - exponentielle de X

* FIX(X)

  - permet de supprimer la partie décimale de X ; exemple : FIX(PI) = 3 où PI = 3.14159

* FRE { (0) / (X$) }

  - elle donne la taille disponible en octets de la mémoire centrale

* HEX$(X)

  - convertit en une chaîne de caractères le nombre X

* INKEY$

  - indique quel a été le dernier caractère frappé sans l'afficher

* INSTR((I),X$,Y$)

  - elle permet de repérer la première apparition de la chaîne Y$ dans la chaîne X$; exemple INSTR ("0122","2") = 3

* INT(X)

  - donne la partie entière de X

* LEFT$(X$,I)

- tronque la chaîne X$ à partir du Iième caractère ; exemple : LEFT$ ("ABCD",2) = "AB"

* LEN(X$)

- donne la longueur de la chaîne X$; exemple : LEN("1234") = 4

* LOG(X)

- donne le logarithme népérien de X

* MID$

Syntaxe : MID$ ([variable chaîne de caractères], n(,m))

n et m sont des expressions numériques

- Elle permet d'extraire ou de remplacer dans la variable chaîne de caractères

  . les caractères numéro n, n + 1, n + 2,...,n + m

  . tous les caractères situés après le nième inclus) au cas où m est omis

  Exemple :

```
10  A$ = "HYDROGLISSEUR"
20  A$ =MID$(A$,5)+"AVION"
30  PRINT A$
40  PRINT MID$ (A$,2,6)
```

Ce programme donne

HYDRAVION puis
YDRAVI

* OCT$(X)

  - convertit X en une chaîne de caractères

* POS ⟨([numéro de canal])⟩

  - elle indique la position du curseur dans le fichier relié au TO7 par le canal indiqué (ou le canal 0 si le numéro n'est pas mentionné) cf § 3.5.

* RIGHT$(X$,I)

  - extrait de la chaîne X$ les I derniers caractères ; exemple RIGHT$ ("ABCD",3) = "BCD"

* RND(X)

  - donne un nombre réel aléatoire entre 0 et 1 exclus

* SGN(X)

  - donne le signe de X

* SIN(X)

  - donne le sinus de X

* SPC(X)

  - écrit X blancs

* SQR(X)

  - racine carrée de X

* STR$(X)

  - convertit X en une chaîne de caractères exemple : STR$(15) = "15"

* TAN(X)

  - tangente de X

* VAL(X$)

  - convertit la chaîne de caractères en une valeur numérique ; exemple : VAL ("36") = 36

Ces fonctions ne sont pas fondamentales. Elles doivent être approfondies au fur et à mesure des besoins que le programmeur pourra ressentir lors de l'écriture de programmes.

3.5 ENTRÉES-SORTIES GÉNÉRALISÉES

Les instructions étudiées précédemment permettent le dialogue entre l'utilisateur et l'ordinateur (en fait entre le clavier et l'unité centrale). Pour disposer de possibilités plus nombreuses, il est nécessaire de pouvoir établir entre l'ordinateur (l'unité centrale) et des périphériques le dialogue indispensable pour enregistrer des programmes, imprimer des messages sur l'écran, utiliser des imprimantes, des lecteurs de disquettes etc... Pour permettre ces dialogues, il faut disposer de voies de communication ou canaux ; il y en a dix-sept sur le TO7.

Parmi ceux-ci :

- Le canal 0 est ouvert en permanence et de manière automatique. C'est par son intermédiaire que l'utilisateur peut dialoguer directement avec l'ordinateur et que ce dernier peut dialoguer avec l'écran pour l'affichage.

- Les seize autres canaux s'ouvrent sur demande.

Remarque : les instructions précédemment étudiées utilisent le canal 0.

3.5.1 Ouverture - fermeture

Pour les autres dialogues possibles entre l'ordinateur et les périphériques il faut que l'utilisateur ouvre des canaux. Un canal par type de dialogue est nécessaire.

Exemples :

Un canal pour le dialogue ordinateur voie série
Un canal pour le dialogue ordinateur voie série

Cependant, pour faciliter l'utilisation des périphériques, l'ouverture d'un canal se fait automatiquement pour certaines opérations (la fermeture étant opérée systématiquement dès la fin des opérations). Ce cas se présente pour tout ce qui concerne des programmes avec le lecteur-enregistreur (pour sauvegarder ou charger (i.e. enregistrer ou lire) un programme) ; avec l'imprimante (pour lister des programmes) ; etc...

Par contre pour enregistrer ou lire des données (en utilisant le lecteur-enregistreur, la voie série, etc...), il est nécessaire d'ouvrir les canaux et de les refermer sitôt l'opération effectuée en utilisant les instructions OPEN CLOSE.

* OPEN- CLOSE

Syntaxes :

```
OPEN {"O"},‹#›[numéro de canal],[descripteur
     {"I"}         de fichier]

CLOSE ‹‹#›[numéro de canal]›...
```

- Pour OPEN

a) Le choix entre O et I se fait de la manière suivante, si le canal doit transmettre des informations :

. de l'ordinateur vers le périphérique, il faut choisir "O" (Output)

. du périphérique vers l'ordinateur, il faut choisir "I" (Input).

b) Le numéro de canal est à choisir entre 1 et 16. Il doit être utilisé autant d'instructions Open qu'il y a de canaux à avoir.

c) Le terme [descripteur de fichier] contient les informations relatives au périphérique, à la nature de transmission, etc... Il doit contenir en général :

. le type de périphérique suivi de deux points, à savoir :

KYBD : (pour le clavier)

SCRN : (pour l'écran)

LPRT : (pour l'imprimante)

COMM : (pour la voie série)

CASS : (pour le lecteur-enregistreur ; qui peut être omis, cf remarque)

Remarque : si le type de périphérique n'est pas précisé dans le descripteur de fichier, le TO7 considère d'autorité le lecteur-enregistreur.

. une "constante" numérique qui, placée entre parenthèses, détermine la manière selon laquelle les messages seront transmis. Elle ne doit être introduite que pour une imprimante ou la voie série. L'imprimante n'est concernée que par l'alinéas 3 ci-après. Seule le voie série exige l'application des trois alinéas ci-dessous. Cette constante s'obtient comme suit :

1. le premier chiffre détermine la vitesse de transmission ; il est à choisir entre 1, 2, 3, 4, 5 et 6 :

(premier chiffre)	(vitesse correspondante)
1	110 bauds
2	300 "
3	600 "
4	1200 "
5	2400 "
6	4800 "

(1 baud est égal à un bit par seconde)

2. le deuxième chiffre détermine le nombre de bits transmis pour un caractère ; il est égal soit à 7 (transmission sur 7 bits) soit à 8 (transmission sur 8 bits)

3. les chiffres suivants représentent le nombre de caractères par ligne (compris entre 0 et 255)

Exemple :

(57128) (transmission sur 7 bits de lignes de 128 caractères à 2400 bauds).

Remarque :

- Cette constante peut être omise ; dans ce cas le TO7 prendra d'autorité la valeur 5740.

. Le nom de fichier sous lequel vont être rassemblées les informations transmises. Il est constitué par :

- un nom qui comporte de un à huit caractères à l'exception de la virgule, du 0, du nombre 255 et du point ;

- et un suffixe optionnel séparé du nom par un point. Ce suffixe comporte de 1 à 3 caractères répondant aux mêmes restrictions que celles relatives au nom (cf ci-dessus).

Remarque : si le suffixe n'est pas précisé le TO7 en ajoute un, selon le cas :

DAT s'il s'agit de données
BIN s'il s'agit d'un fichier binaire
BAS s'il s'agit d'un fichier programme

N.B. : le descripteur de fichier est à mettre entre guillemets.

Exemples :

1. Descripteurs de fichier

"COMM : (57255) ESSAI.ONE"
"PGM.BON" (ce dernier est équivalent à "CASS : PGM.BON")

2. Utilisation de l'instruction Open

OPEN "I", # 3, "COMM:(57255) ESSAI.TWO"

Cette instruction ouvre le canal 3 en I(nput) pour communiquer avec la voie série sur le fichier ESSAI.TWO : avec une transmission sur 7 bits à 2400 bauds, le nombre de caractères par ligne étant 255,

Remarque : si vous essayez d'ouvrir un canal déjà ouvert, le message suivant apparaît :

? AO Error (AO est mis pour Already Open)

- L'instruction CLOSE ferme les canaux indiqués. Les différents numéros de canaux doivent être séparés les uns des autres par une virgule (ils sont entiers et compris entre 1 et 16).

Exemple :

CLOSE 1, 3, 15

La demande de fermeture d'un canal déjà fermé n'entraîne pas de message d'erreur. Par contre un numéro de canal incorrect provoque l'affichage suivant :

? NU Error (NU est mis pour Not in Use)

Les canaux inutilisés doivent être fermés.

3.5.2 Entrée de données

* INSTRUCTIONS INPUT#, LINEINPUT # et INPUT$

Syntaxes :

INPUT#[numéro de canal] , [variable] ⟨,[variable]⟩

LINEINPUT#[numéro de canal] , [variable chaîne]

INPUT$([A] <,<#> [numéro de canalB]>)

Remarque préliminaire : pour ces instructions il faut nécessairement qu'une ouverture de canal ait été effectuée avant de les utiliser ; sans quoi l'erreur suivante est détectée :

? NO Error (NO pour Not Open)

Cette ouverture doit être faite en Input.

L'étude portera successivement sur les deux premières puis sur la troisième.

* **INPUT # LINE INPUT #** (avec ou sans blanc entre (LINE INPUT#) LINE et INPUT)

Elles s'utilisent de la même manière qu'INPUT et LINEINPUT, mais il ne faut pas oublier le symbole # suivi du numéro de canal.

Exemple 1 :

```
10  OPEN "I",#1, "LYCEE"
20  INPUT # 1, NOM$, NOTE, CLASSEMENT
30  PRINT NOM$ ; ":" ; NOTE ; ":" ;
    CLASSEMENT
40  CLOSE 1
```

Ce programme permet de lire dans le fichier LYCEE (inscrit sur cassette) une chaîne de caractères, et deux valeurs numériques qui vont respectivement être placées dans NOM$, NOTE et CLASSEMENT (ligne 20).

Exemple 2 :

```
10  OPEN "I", #1, "KYBD : NOMS"
20  LINE INPUT #1, NOM$
30  PRINT NOM$
40  CLOSE  1
```

Ce programme "permet" le dialogue entre le clavier et l'ordinateur (soit entre l'utilisateur et l'ordinateur) par le moyen du canal 1.

Remarques :

. Le symbole # n'est pas obligatoire pour OPEN et CLOSE mais il l'est pour INPUT # et LINE-INPUT #.

. L'ouverture de canaux entre le lecteur-enregistreur et l'ordinateur nécessite la commutation :

- de la touche lecture pour une ouverture en Input ;

- des touches lecture et enregistrement pour une ouverture en Output.

* **INPUT$** (nombre, canal)

Exemple :

```
10  OPEN "I", 3, "NOMS"
20  NOM$ = INPUT$ (5,  3)
30  CLOSE  3
```

La ligne 10 ouvre le canal 3 en Input pour accéder au fichier NOMS inscrit sur cassette.

La ligne 20 permet de récupérer les cinq premiers caractères figurant dans ce fichier (NOMS ; car le canal 3 est ouvert sur ce fichier ; cf ci-avant).

3.5.3 Fonction EOF (End of File)

Syntaxe :

EOF ([numéro de canal])

Lorsque des canaux sont ouverts en Input il est utile de pouvoir déterminer s'il reste des informations dans le fichier avec lequel l'unité centrale dialogue. Ceci permet d'éviter de lire un fichier vide ou vidé (par suite de lectures successives), ce qui conduirait au message d'erreur ? IE Error. EOF réalise cette détermination.

Elle renvoie la valeur :

- vrai (-1) si le fichier est vide (ou vidé)

- faux (0) sinon

Cette instruction est toujours utilisée avec une instruction IF - THEN (ELSE).

Exemple :

```
10  OPEN "I", 15, "DONNEES"
20  IF EOF(15) THEN 60
30  INPUT#15, DONNEE
40  PRINT "DONNEE"
50  END
60  PRINT "FICHIER DONNEES VIDE"
70  END
```

Ce programme permet de lire dans le fichier DONNEES, situé sur cassette, à travers le canal 15. Si ce fichier contient un élément l'ordinateur l'imprime, sinon il écrit FICHIER DONNEES VIDE.

3.5.4 Sortie des données

PRINT # PRINT#USING

Syntaxes :

PRINT # [numéro de canal],([expression])

PRINT # [numéro de canal],USING [chaîne image] ;([expression])...

La différence entre ces deux instructions et les deux précédentes est la même que celle entre INPUT et INPUT #. Il s'agit de l'utilisation d'un canal différent du canal 0 pour pouvoir par exemple :

- enregistrer des données dans un fichier sur cassette ;
- enregistrer des données dans un fichier dans la voie série.

Pour ces deux instructions, il ne faut pas oublier :

- d'avoir un canal en Output avant de les utiliser sans quoi l'erreur ? NO Error apparaît (NO pour Not Open) ;
- de refermer le canal dès qu'il n'est plus utilisé.

Remarque : avec l'utilisation du lecteur-enregistreur, il ne faut pas oublier d'enfoncer simultanément les touches lecture et enregistrement.

Exemple :

Pour enregistrer des données, rentrées au clavier, sur cassette, on peut partir du programme suivant :

```
10  OPEN "O", 1, "DONNEES"
20  INPUT "VALEUR" ; A
30  PRINT # 1, A
40  INPUT "VOULEZ-VOUS ENREGISTRER D'AUTRES
    DONNEES", REPONSE$
50  IF REPONSE$ <>"OUI" THEN 60 ELSE GOTO 20
60  CLOSE 1
70  END
```

CHAPITRE **4**

Le graphisme du TO 7

4.1 INTRODUCTION

Les différentes instructions BASIC décrites jusqu'ici vous permettent de mener à bien des calculs plus ou moins complexes, de traiter des chaînes de caractères ou encore de créer des jeux de réflexion. Cependant, votre ordinateur vous communique ses résultats uniquement sous forme de suites de caractères alphanumériques, à l'exclusion de toute forme d'image ou dessin et encore bien moins sous forme d'animations. Utilisé ainsi, votre écran de télévision couleur ne présente pas beaucoup plus d'interêt qu'un terminal classique, c'est à dire uniquement doté de l'affichage de signes, lettres et chiffres identiques à ceux du clavier.

Le jeu d'instructions décrit dans ce chapitre apporte une dimension nouvelle à vos programmes BASIC sur TO7. En effet, ces instructions vous conduiront dans le domaine passionnant de la micro-informatique vidéo.

Nous allons bien entendu passer en revue toutes les possibilités graphiques de votre TO7, mais auparavant, pour bien utiliser le graphisme, il est nécessaire de comprendre la manière dont une image est organisée et représentée.

4.2 REPRÉSENTATION DES IMAGES SUR TO 7 ET MODE GRAPHIQUE

L'image provenant du TO7 et apparaissant sur l'écran de télévision est formée d'un cadre (dans lequel il est impossible d'écrire mais dont la couleur est modifiable) et de la fenêtre de travail. Par abus de langage, nous appellerons par la suite "Image" cette fenêtre de travail puisqu'elle constitue la seule partie de l'écran nous intéressant réellement.

4.2.1 Généralités

L'image fournie par le TO7 peut être considérée comme formée de 200 lignes, chaque ligne se divisant en 40 zones ou secteurs de 8 points consécutifs (voir figure ci-dessous), ceci étant dû au fait que la mémoire d'image est organisée en octets de points.

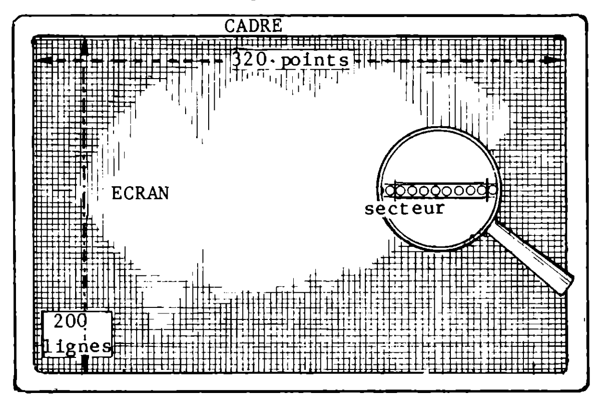


FIGURE 1 : L'ECRAN DE TELEVISION

Nous voyons apparaître à ce stade une restriction essentielle aux machines appartenant à cette gamme de prix. En effet, il n'est pas possible de colorer chaque point d'une image de façon totalement indépendante du secteur de 8 points alignés dont il fait partie.

Cependant cette petite restriction, grâce à laquelle le graphisme couleur sur ordinateur peut être mis à la disposition du grand public, ne gêne pas la très grande majorité des programmes que l'on peut être conduit à écrire.

Chaque secteur de 8 points ne peut donc posséder que 2 couleurs : une couleur de fond et une couleur de graphisme. Par contre, la paire de couleurs graphisme-fond d'un secteur quelconque est totalement indépendante des autres secteurs de l'écran.

Ainsi, une image est constituée de 200 lignes comportant chacune 320 points. L'ensemble de ces points convenablement colorés forment l'image désirée. C'est pour cette raison que l'on parle à propos du TO7 d'une résolution graphique de 64 000 points ou encore : 200 x 320.

4.2.2 Exemples de possibilités graphiques

Pour faire apparaître sur l'écran une image suivant le seul principe décrit ci-dessus, il faudrait donc attribuer aux 200 x 40 = 8 000 secteurs de 8 points une couleur de graphisme et une couleur de fond et, pour chacun des 8 points d'un secteur, préciser s'il doit posséder la couleur du graphisme ou celle du fond. La synthèse d'une telle image, bien que possible sur TO7, serait longue et fastidieuse à mettre en oeuvre.

En fait, des instructions spécialement étudiées sont à votre disposition et permettent de réduire la plupart des programmes de dessin à quelques lignes. Par exemple, pour dessiner un rectangle mauve entouré de bleu foncé et écrire à l'intérieur "BONJOUR" en caractères de taille double et de couleur rouge, le tout sur fond noir, le programme suivant suffit :

```
10  SCREEN 5, 0, 0 : CLS
20  BOXF (11,7)-(30,16) CHR$ (127)
30  BOX (10,6)-(31,17) CHR$ (127), 4
40  ATTRB 1, 1 : COLOR 1, 5 : LOCATE 14, 12
50  PRINT "BONJOUR" : ATTRB 0, 0 : END
```

Après exécution du programme, pour revenir aux couleurs de départ, il faut successivement exécuter :

[RAZ] puis SCREEN 4, 6, 6 et [ENTREE]

4.2.3 Visualisation des secteurs et du cadre

Dans le but de faire visualiser aux lecteurs quelques-uns des 8 000 secteurs que comporte un écran géré par TO7 nous proposons la manipulation suivante:

Tout d'abord, afin de mieux distinguer les segments de droite que forment les secteurs, passons en affichage vert sur fond noir en écrivant :

SCREEN 2, 0, 0 [ENTREE]

puis appuyons sur les touches [ENTREE] et [RAZ]. Le curseur que l'on peut voir clignoter constitue lui-même un des 8 000 secteurs de l'image ; mais faisons apparaître d'autres secteurs ; pour cela, entrons :

POKE 20403, 255 [ENTREE]

Un segment de droite vert apparaît au beau milieu de l'écran ; il représente le 4 020ième secteur de l'écran (compté de gauche à droite et ligne par ligne). Vous pouvez constater qu'il est bien formé de 8 petits points consécutifs. En entrant maintenant :

```
POKE 20323, 255 [ENTREE]
```

un autre secteur apparaît 2 lignes au-dessus du précédent.

De façon générale, pour allumer le Nième segment, il faut entrer :

```
POKE 16383 + N, 255 [ENTREE]
```

avec : N = 40 * L + 5 ; L représentant le nombre de lignes et H l'ordre du secteur dans cette ligne.
Cette façon de procéder sera expliquée au paragraphe consacré à la constitution physique de la mémoire d'écran.

Le programme suivant permet de visualiser sur votre écran le cadre de couleur rouge ainsi que les lignes dont les secteurs auront alternativement une couleur noire ou blanche.

```
10  SCREEN 2, 0, 1 : C = 0 : CLS
20  FOR J = 0 TO 199
30  C = (C = 0) * 8
40  FOR I = 0 TO 319 STEP 8
50  PSET (I,J), C : C = (C = 0) * 8
60  NEXT I : NEXT J : END
```

Après avoir exécuté le programme il faudra, pour revenir aux couleurs d'origine, entrer successivement:

```
[RAZ]
SCREEN 4, 6, 6 [ENTREE]
```

4.2.4 La couleur

Maintenant que nous commençons à mieux concevoir la façon dont est structurée une image graphique, il est temps de compléter ces premières notions par la plus intéressante : la couleur.

Le système TO7 permet de générer sur l'écran 8 couleurs qui seront codées dans la plupart des instructions par 8 chiffres de 0 à 7. Le tableau suivant donne ces huit couleurs ainsi que le code correspondant à chacune d'entre elles.

couleur	code graphisme ou fond	code fond
NOIR	0	- 1
ROUGE	1	- 2
VERT	2	- 3
JAUNE	3	- 4
BLEU (bleu foncé)	4	- 5
MAGENTA (violet)	5	- 6
CYAN (bleu clair)	6	- 7
BLANC	7	- 8

La signification de ces codes apparaîtra plus clairement par la suite.

Sans plus attendre, voici un programme permettant d'apprécier les différentes couleurs sur différents fonds. Un rectangle de petite dimension prend successivement les 8 couleurs possibles à l'intérieur d'un rectangle plus grand qui change de couleur huit fois moins rapidement que le petit de façon à ce que toutes les possibilités soient visualisées.

```
10  SCREEN 2, 0, 0 : CLS
20  FOR I = 1 TO 7
30  FOR J = 0 TO 7
40  BOXF  (84,56)-(236,144), I
50  BOXF  (140,88)-(180,112), J
60  FOR K = 1 TO 500 : NEXT K
70  NEXT J : NEXT I : END
```

Après l'exécution de ce programme, l'écran écrit en caractères verts sur fond noir. Cette présentation est à la fois reposante pour les yeux et pour votre téléviseur ; en effet, seuls les caractères utiles étant allumés, le tube cathodique de l'écran du téléviseur aura une usure plus lente qu'avec la présentation bleu foncé sur fond bleu clair. Pour cette raison, nous conseillons au lecteur de travailler avec un écran ainsi coloré. Le passage dans ce mode de couleur sera effectué par la commande :

SCREEN 2, 0, 0 [ENTREE]

4.2.5 Visualisation des couleurs dans un secteur

Nous allons visualiser ici tout comme au paragraphe 2.3 ce qui a été dit au sujet des secteurs noirs, dans le domaine de la couleur.

Intéressons-nous au secteur 4 020. Dans un premier temps, nous allons le faire apparaître avec ses quatre premiers points en vert (couleur de graphisme) et les quatre derniers en noir (couleur de fond). Pour cela, il faut effectuer dans l'ordre :

SCREEN 2, 0, 0 si vous ne l'avez déjà fait
[RAZ]
POKE 20403, 240 [ENTREE]

Comme prévu, seuls les quatre premiers points du secteur sont visibles. Entrons maintenant le programme suivant (après avoir fait un NEW) :

```
10  POKE &HE7C3, 1
20  POKE 20403, 240
30  POKE &HE7C3, 0
40  POKE 20403, 12
50  END
```

Puis exécutons-le après avoir fait un RAZ. Le secteur 4 020 apparaît alors, mais cette fois-ci les quatre premiers points sont de couleur rouge et les quatre derniers de couleur bleue. Jusque là, tout se passe comme prévu.

Essayons maintenant de colorer en vert le premier point du secteur (initialement de couleur rouge). Pour cela, effectuons la commande suivante :

```
PSET (152, 100), 2 [ENTREE]
```

Nous constatons alors que tous les points rouges qui appartenaient dans ce cas au graphisme sont devenus verts. Une constatation s'impose : il est impossible de colorer un secteur à l'aide de plus de deux couleurs ; ceci est normal en vertu de tout ce qui a été énoncé au début du présent chapitre. En changeant le chiffre 2 dans la commande précédente par un chiffre quelconque entre 0 et 7, l'ensemble des quatre premiers points revêtira la couleur correspondante.

Si maintenant, nous essayons de changer le fond en entrant :

soit   PSET (158,100), - 4 [ENTREE]

soit   PSET (158,100), - 6 [ENTREE]

soit   PSET (158,100), - 7 [ENTREE]

nous vérifions qu'il n'est possible de changer que deux couleurs.

Cependant, il est important à ce stade d'avoir bien compris que dans un secteur donné, les points ne peuvent prendre que deux couleurs, mais que n'importe quel point peut prendre la couleur du fond ou celle du graphisme.

Pour finir cette illustration, exécutons à nouveau le programme précédent (RUN) après avoir fait un RAZ ; puis entrons :

```
POKE 20404, 255 [ENTREE]
```

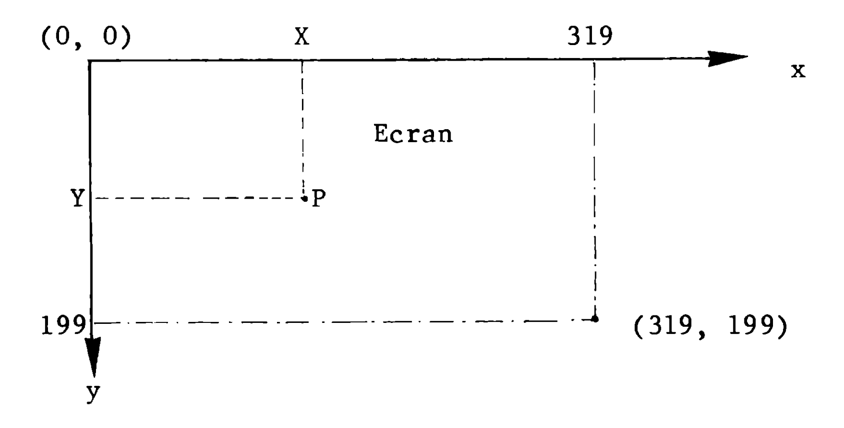
Le secteur 2 041 apparaît à la suite du 2 040, nous voyons que ces deux secteurs consécutifs possèdent des couleurs totalement indépendantes. Ceci confirme que deux secteurs peuvent être colorés de façon totalement indépendante.

4.2.6 Mode graphique et coordonnées d'un point sur l'écran

Lorsque l'on manipule des points sur l'écran, qu'on leur attribue des couleurs, qu'on les regroupe en formes quelconques, on dira que l'on est en mode graphique et que l'on utilise des instructions graphiques. Ceci par opposition au mode caractère que l'on utilise pour inscrire sur l'écran des paquets de points qui représentent des signes connus et utilisés de façon fréquente.

Les instructions graphiques qui permettent de positionner et de colorer les points ou les groupes de points nécessitent bien entendu une désignation précise de la position des points concernés. A cette fin, un système de coordonnées très simple a été standardisé pour tous les BASIC Microsoft.

Un point P quelconque de l'écran est repéré par ses coordonnées (X, Y). Y représentant le nombre de lignes à compter en partant du haut de l'écran (ligne 0) pour atteindre le point P ($0 \leqslant Y \leqslant 199$) et X le nombre de points qu'il faut compter sur la ligne Y en partant de la gauche de l'écran (point 0) pour atteindre P ($0 \leqslant X \leqslant 319$). Ainsi, le point situé le plus en haut à gauche de l'écran a pour coordonnées (0, 0) et le plus en bas à droite (319, 119). Remarquons que de 0 à 199, nous avons bien 200 lignes et de 0 à 319 nous comptons bien 320 points.



Ces coordonnées serviront à spécifier le point sur lequel on voudra faire agir l'instruction graphique. Par exemple pour positionner un point rouge au centre de l'écran (160, 100) il suffira d'écrire :

```
PSET (160, 100), 1 [ENTREE]
```

Les coordonnées du point considéré apparaissent clairement dans cette instruction. D'autres instructions nécessitent deux couples de coordonnées.

Par exemple, pour tracer une ligne rouge entre le point A (100, 150) et le point B (300, 190) il faudra utiliser l'instruction

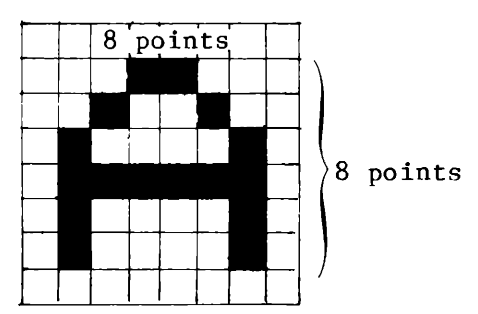
LINE (100, 150) - (300, 190), 1 [ENTREE]

Là encore, les coordonnées des points A et B sont présentes dans l'instruction.

4.3 REPRÉSENTATION DES CARACTÈRES SUR TO 7 ET MODE CARACTÈRE

4.3.1 Définition d'un caractère

Un caractère est un bloc carré de 64 points (8 x 8). La forme du caractère est définie par l'appartenance de chacun des 64 points au fond ou au graphisme. Par exemple, le caractère "A" est formé par les 16 points ombrés appartenant au graphisme et les 48 points non colorés appartenant au fond dans le dessin ci-dessous :



Bien entendu, il n'est pas utile à chaque fois que l'on veut faire apparaître un caractère sur l'écran de sélectionner les 64 points qui devront appartenir au graphisme ou au fond. Ceci est dû au fait que chaque caractère a été mémorisé par construction dans votre TO7.

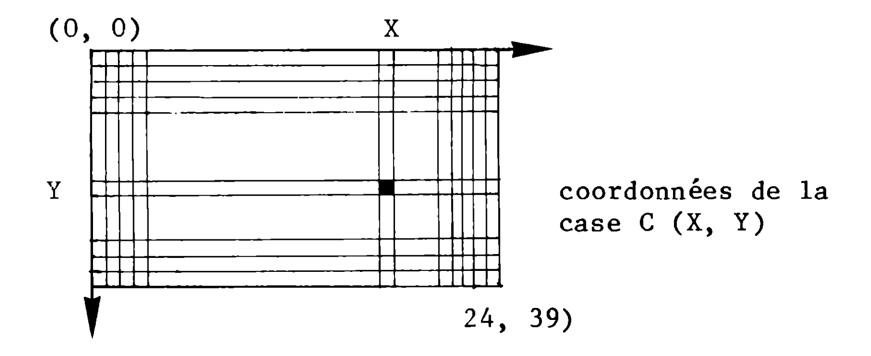
Ainsi, le programme qui gère votre ordinateur prend en charge cette opération ; chaque fois que l'écriture d'un caractère sur l'écran lui est demandée, il affiche les 64 points conformément à la répartition préprogrammée en mémoire.

Afin de faire un lien avec tout ce qui a été vu à propos de la représentation des images sur TO7, il faut retenir qu'un caractère coïncide toujours sur l'écran avec 8 secteurs empilés. Le programme de gestion de l'ordinateur n'affiche donc pas véritablement les 64 points les uns après les autres mais affiche les 8 secteurs correspondants, chaque secteur ayant reçu sa répartition de points fond et graphisme. D'autre part, en raison de cette disposition en secteurs de l'image, les caractères ne peuvent occuper que des cases de 64 points bien définies sur l'écran. C'est ainsi que nous sommes menés à la notion de coordonnées en mode caractère.

4.3.2 Coordonnées en mode caractère

En ce qui concerne le mode caractère, l'écran peut être considéré comme constitué de 25 lignes, chaque ligne étant divisée en 40 cases dans lesquelles on pourra inscrire les caractères. Ces cases comportent bien évidemment 64 points en matrice 8 x 8 et l'intérêt du mode caractères est, entre autres choses, d'éviter d'avoir à calculer la position de ces 64 points sur l'écran lorsque l'on veut inscrire un caractère dans une case quelconque.

Tout comme pour le mode graphique, on attribue à chaque case C un couple de coordonnées (X, Y). X désignera le nombre de cases en partant de la gauche de l'écran (case 0) que l'on devra compter pour atteindre la case C, et Y le numéro de la ligne sur laquelle la case C se situe (la ligne 0 étant en haut de l'écran) ($0 \leqslant X \leqslant 39$) et $0 \leqslant Y \leqslant 24$).



Les cases extrêmes ont pour coordonnées (0 0) et (24, 39). On pourrait aussi parler de lignes pour Y et de colonnes pour X.

Les coordonnées serviront à spécifier la case sur laquelle on voudra faire agir l'instruction de caractère. Par exemple pour positionner un caractère "A" au milieu de l'écran (colonne 19 et ligne 12) il faudra écrire :

PSET (19, 12) "A" [ENTREE]

Les coordonnées (X, Y) apparaissent clairement dans cette instruction que nous verrons plus en détail par la suite.

Le programme suivant permet de visualiser l'ensemble des 40 x 25 = 1 000 cases de 64 points du mode caractère. Le cadre apparaît en rouge et les cases alternativement en vert et en noir. La case de coordonnées (19, 12) s'inscrit en rouge sur l'écran.

```
10  SCREEN 2, 0, 1 : C = 2 : CLS
20  FOR J = 0 TO 24
30  C = - (C = 0) * 2
40  FOR I = 0 TO 39
50  PSET (I,J), CHR$ (127), C
60  C = - (C = 0) * 2
70  NEXT I : NEXT J
80  PSET (19, 12), CHR$ (127), 1
90  COLOR 2 : END
```

Après avoir vérifié que la case rouge a bien pour coordonnées (19, 12), vous reviendrez en mode normal en exécutant les opérations suivantes :

```
[RAZ]
SCREEN 4, 6, 6 [ENTREE]
```

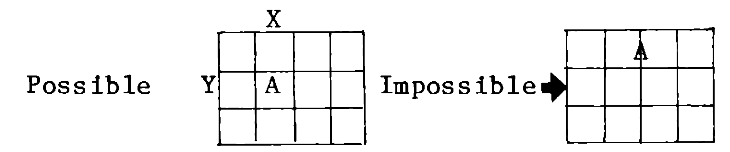
4.3.3 Les caractères ASCII

Physiquement, les informations véhiculées et manipulées par un ordinateur sont une suite de 0 et de 1. Un ensemble de 0 et de 1 ordonnés peut être assimilé à un nombre. C'est la raison pour laquelle, pour manipuler une information quelconque (lettres, signes, mots, dessins, etc...), il est nécessaire d'établir une correspondance rigoureuse entre l'information traitée et un nombre qui la représentera dans la machine. Cette correspondance s'appelle un codage. Ainsi, il existe une multitude de codes qui peuvent être utilisés pour représenter par exemple l'alphabet ; le plus simple étant A = 1, B = 2, ..., Z = 26.

Dans un ordinateur, il y a beaucoup plus de caractères et autres informations que les seules lettres de l'alphabet, ne serait-ce que tous les caractères que comporte votre clavier.

Afin d'ordonner l'informatique dans ce domaine, un Code Standard a été mis au point et adopté par la majorité des concepteurs ; il s'agit du code ASCII (American Standard Code for Information Interchange).

Le code ASCII est utilisé par certaines instructions, il faut savoir l'employer correctement pour profiter pleinement des capacités graphiques de votre TO7. Il est important de savoir que le code ASCII comprend en plus des signes qui apparaissent sur l'écran des caractères de commande tels que : le déplacement du curseur, l'effacement ou le beep ; ces caractères pourront être utilisés dans des programmes d'animation. Bien entendu, tous les caractères du code ASCII qui ne sont pas des caractères de commande sont transformés en matrice 8 x 8 (blocs de 64 points) lors de leur affichage et placés dans la case de coordonnées spécifiées ; comme tout caractère, un caractère ASCII ne peut chevaucher deux ou plusieurs cases de 64 points.



4.3.4 Les caractères programmables

Malgré tous les caractères que possède le TO7 dans sa mémoire ROM (non effaçable), l'utilisateur est souvent amené à utiliser ses propres caractères, surtout dans des programmes de jeu. Une instruction spéciale : DEFGR$ lui permet de les définir et donc, de créer son propre jeu de caractères associé à son propre code.

L'utilisation de l'instruction DEFGR$ sera expliquée en détail au paragraphe 4.5. Cependant, nous pouvons déjà expliquer la façon dont un caractère est défini pour cette instruction.

Comme il a déjà été dit au paragraphe 3.1, un caractère est formé de 8 secteurs placés les uns au-dessus des autres, le tout formant un bloc carré de 64 points (8 x 8). Chaque secteur étant formé de 8 points, il est possible de le définir à l'aide d'un octet (nombre binaire composé de 8 chiffres binaires (1 ou 0). Chaque chiffre binaire prenant le nom de bit, un octet est composé de 8 bits. Ces bits sont numérotés de 7 à 0 ; le plus à gauche étant le bit 7 et le plus à droite le bit 0. La correspondance entre les bits d'un octet et un secteur est très simple : par exemple, si le point 6 du secteur appartient au fond, alors le bit 6 de l'octet vaudra 0 dans le cas contraire, bien entendu, il vaudra 1 (le point 6 étant le deuxième point du secteur en partant de la gauche).

Prenons pour illustrer ceci le secteur suivant où les points sombres représentent les points graphisme.

● ○ ● ● ○ ○ ○ ●

L'octet définissant ce secteur aura pour valeur :

1 0 1 1 0 0 0 1

qui est une valeur binaire. Cette valeur peut être convertie en valeur décimale :

$1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 177$

Ainsi, 177 sera la représentation du secteur décrit.

Un caractère étant formé de 8 secteurs, il sera décrit par une suite de 8 nombres déterminés par la même méthode que précédemment. Cette suite commencera par le nombre représentant le secteur le plus haut du caractère et finira par le secteur le plus bas, les nombres étant séparés par une virgule.

Par exemple, pour décrire le caractère suivant :

	128	64	32	16	8	4	2	1	
1e secteur	●		●	●				●	177
2e secteur	●		●	●			●	●	179
3e secteur	●		●	●		●	●	●	183
4e secteur	●	●	●	●	●	●	●		254
5e secteur	●	●	●	●	●	●	●		254
6e secteur	●		●	●		●	●	●	183
7e secteur	●		●	●			●	●	179
8e secteur	●		●	●				●	177

8 points (vertical) — 8 points (horizontal)

on utilise la suite de nombres 177, 179, 183, 254, 254, 183, 179, 177. Un moyen pratique pour calculer la valeur d'un secteur est d'ajouter tous les nombres situés en haut du carré dont le point dans la colonne correspondante appartient au graphisme (car 2^7 = 128, 2^6 = 64 etc...). Ainsi le nombre 177 a été déterminé par l'opération 128 + 32 + 16 + 1 = 177.

Ce caractère pourra être visualisé sur l'écran en exécutant le programme :

```
10  CLEAR, , 1
20  DEFGR$ (0) = 177, 179, 183, 254, 254,
    183, 179, 177
30  PRINT GR$ (0) : END
```

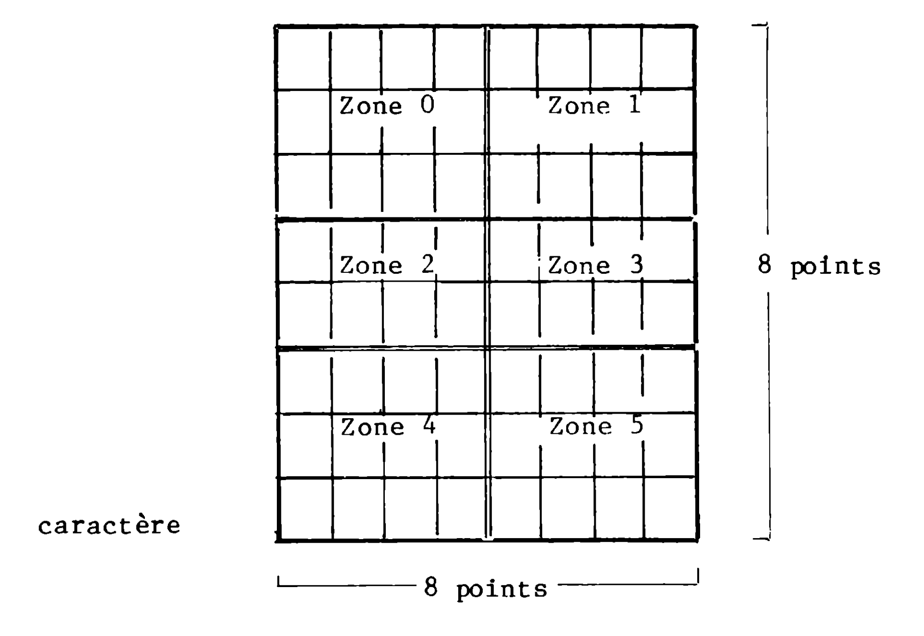
La suite de nombres définissant le caractère graphique désiré apparaît clairement à la ligne 20 du programme.

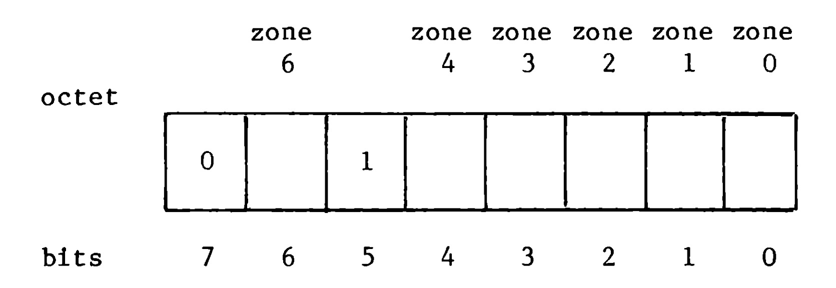
4.3.5 Les caractères semi-graphiques TELETEL

En plus du jeu de caractères ASCII, le TO7 dispose des caractères semi-graphiques TELETEL. Tout comme les caractères ASCII, les caractères TELETEL possèdent un code qui s'utilise avec les mêmes instructions que le code ASCII ; ce code est étroitement lié au dessin des caractères qu'il représente.

Les caractères s'inscrivent évidemment dans une matrice 8 x 8 (case de 64 points) ; ils sont divisés en 6 zones, chaque zone correspondant à un bit dans l'octet du code les représentant. Les deux bits non utilisés (7 et 5) sont respectivement à 0 et à 1. Si une zone fait partie du fond, le bit associé sera à 0 et dans le cas contraire à 1 (graphisme).

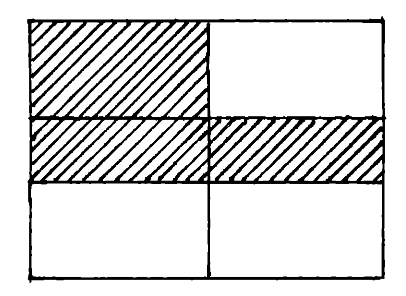
Les 6 zones et l'octet de codage sont indiqués ci-dessous :



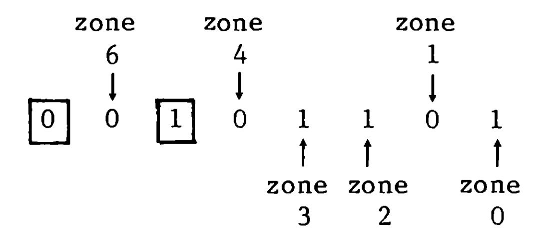


Prenons un exemple :

Soit à trouver le code du caractère semi-graphique TELETEL suivant (la partie sombre représente le graphisme et le blanc le fond) :



Les zones 0, 2 et 3 appartiennent au graphisme et les zones 1, 4 et 6 au fond ; l'octet sera donc :



Ce nombre binaire converti en décimal donne 45 ; 45 est donc le code TELETEL du caractère ci-dessus.

Le passage en mode graphique TELETEL s'effectue par un PRINT CHR$(14) et le retour au mode normal par un PRINT CHR$(15). En mode graphique TELETEL, les caractères sont affichés par un PRINT CHR$(N) où N est le code du caractère désiré.

Exemple 1 :

Affichons sur l'écran le caractère précédemment calculé. Pour cela il suffit de rentrer :

```
PRINT CHR$(14), CHR$(45), CHR$(15) [ENTREE]
```

Exemple 2 :

Nous allons à l'aide de ce jeu de caractères dessiner le profil d'un bateau. Le programme est le suivant :

```
 10  PRINT CHR$(14)
 20  FOR J = 1 TO 2
 30  FOR I = 1 TO 12
 40  READ A
 50  PRINT CHR$(A)
 60  NEXT I : PRINT : NEXT J
 70  PRINT CHR$(43)
 80  FOR I = 1 TO 11
 90  PRINT CHR$(127) ; : NEXT I
100  PRINT CHR$(39)
110  PRINT CHR$(15) : END
120  DATA 32, 32, 32, 32, 32, 32, 53, 32, 32,
     32, 32, 32
130  DATA 32, 40, 124, 32, 32, 127, 127, 124,
     48, 32, 60, 32
```

La ligne 10 fait passer en mode semi-graphique TELETEL et la ligne 110 nous ramène en mode normal.

Le bateau a été dessiné en 3 étages ; le plus haut est décrit par la ligne 120 (uniquement des blancs puis une cheminée) ; l'étage intermédiaire est décrit à la ligne 130. Ces deux étages sont affichés par les lignes 20 à 60. Les lignes 70 à 100 font apparaître le bas du bateau. Les points-virgules utilisés aux lignes 50, 70 et 90 servent à accoller les caractères ensemble. Le bateau est représenté ci-dessous ainsi que la forme et le code de chaque case.

32	32	32	32	32	32	53	32	32	32	32	32	
32	40	124	32	32	127	127	124	48	32	60	32	
43	127	127	127	127	127	127	127	127	127	127	127	39

4.4 CONSTITUTION PHYSIQUE DE LA MÉMOIRE D'ÉCRAN

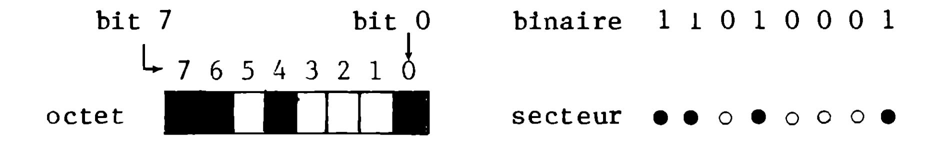
Ce paragraphe peut être ignoré des débutants en micro-informatique, la compréhension de ce qui suit n'en sera pas affectée. Cependant, il permettra à d'éventuels programmeurs en langage d'assemblage d'utiliser leurs connaissances pour accélérer certains jeux et certaines animations. Toutefois, pour ne pas décourager le lecteur désireux d'en savoir plus mais débutant, les explications seront volontairement nombreuses.

La mémoire d'écran se situe de l'adresse &H4000 à &H5F40 , soit en décimal de la case mémoire 16 384 à la case mémoire 16 384 + 8 000 = 24 384. Chaque case mémoire contient un octet (nombre binaire formé de huit chiffres) correspondant à un secteur de l'écran.

Il y a une correspondance directe entre les bits (chiffres binaires) de chaque octet et le graphisme du secteur correspondant. Ainsi, chaque bit de l'octet représente un point du secteur ; le bit 7 étant associé au premier point en partant de la gauche du secteur et le bit 0 au dernier point. Si le bit est à 1, le point appartient au graphisme et si le bit est à 0, le bit appartient au fond.

Exemple :

Si le secteur 39 (première ligne, dernier secteur) est allumé de la façon suivante :



(les points sombres représentent la couleur forme ou graphisme et les points clairs le fond) alors, l'octet mémoire 1 638 + 39 = 16 423 aura pour valeur binaire $(1\ 1\ 0\ 1\ 0\ 0\ 0\ 1)_2$ =&HD1 = 209 (à l'adresse &H 4027).

Il est possible de vérifier ceci en faisant :

POKE 16423, 209 [ENTREE]

ou encore

POKE &H4027, &HD1 en notation hexadécimale

Le secteur apparaît sur la droite de l'écran à l'endroit prévu et avec la forme prévue.

Si l'on numérote les secteurs de gauche à droite et de haut en bas, le premier étant le 0ième et le dernier le 7 999ième, l'octet mémoire correspondant au Nième secteur se trouvera à l'adresse 16 384 + N en décimal et &H4 000 + N en hexadécimal. Nous rappelons qu'il y a 40 secteurs par ligne et 200 lignes.

En fait, la mémoire d'image est constituée de deux plans mémoire. Le premier plan mémoire est celui dont il vient d'être question et représente le plan graphisme puisqu'il détermine les points fond et les points graphisme. Son adressage se situe entre &H4000 et &H5F3F. Le deuxième plan mémoire est le plan couleur, il comporte également 8 000 octets, chaque octet étant attribué au secteur correspondant sur l'écran exactement de la même manière que pour le premier plan mémoire. De plus, ces deux plans se situent exactement à la même adresse, de sorte que l'octet graphisme et l'octet couleur du Nième segment seront tous deux à l'adresse &H4000 + N (ou 16 384 + N en décimal). Une question vient alors à l'esprit : Comment deux octets différents peuvent-ils se trouver à la même adresse ? La réponse est simple, il existe en fait un buffer (certainement un PIA) dont le contenu 1 ou 0 sélectionne respectivement le premier plan mémoire ou le deuxième. Ce buffer se trouve à l'adresse &HE7C3 et est habituellement positionné à 1. Lorsque l'on écrit en mode machine de bureau :

```
POKE 16423, 209 [ENTREE]
```

Le buffer est déjà positionné à 1 ce qui explique que l'on ait toujours accès dans ce cas au premier plan mémoire. Si l'on veut positionner le buffer à 0, il faudra le faire par programme.

Le codage de la couleur graphisme et fond pour un secteur donné se fait par l'intermédiaire de l'octet correspondant dans le deuxième plan mémoire de la façon suivante :

Les bits 7 et 6 sont inutilisés, les bits 5 à 3 contiennent en binaire le code de la couleur graphisme et les bits 2 à 0 le code de la couleur de fond. Le code est celui indiqué au paragraphe 2.4 du présent chapitre.

Exemple :

Nous voulons faire apparaître le secteur 39 en rouge sur fond jaune ; le code du rouge est 1 soit 001 en binaire ; le code du jaune est 3 soit 011 en binaire. L'octet correspondant (adresse 16 423) contiendra :

X X 0 0 1 0 1 1

forme fond

X X signifie que ces valeurs peuvent être quelconques ; 001 représente le code graphisme et 011 le code fond. Prenons pour X X : 0 0 ; alors l'octet est $(0\ 0\ 0\ 0\ 1\ 0\ 1\ 1)_2$ = &H0B = 11

Si nous prenons la même disposition graphisme-fond que dans l'exemple précédent, le programme vérifiant tout ce qui a été dit s'écrit :

```
10  POKE &HE7C3, 1
20  POKE 16423, 209
30  POKE &HE7C3, 0
40  POKE 16423, 11
```

- la ligne 10 sélectionne le 1er plan mémoire ;
- la ligne 20 met dans l'octet graphisme du secteur 39 la bonne configuration ;
- la ligne 30 sélectionne le 2ème plan mémoire ;
- la ligne 40 met dans l'octet couleur du secteur 39 les couleurs voulues.

Après un RUN, on vérifie que le secteur s'est allumé comme prévu.

A ce stade de nos observations, deux vérités énoncées a priori dans les paragraphes précédents s'expliquent :

- La première concerne le fait qu'il est impossible de colorer indépendamment les 8 points d'un même secteur puisque leurs couleurs sont définies par le même octet et que ce dernier ne peut contenir que l'information correspondant à deux couleurs.

- La deuxième explique le codage des couleurs. En effet, nous savons tous qu'une image est constituée de trois couleurs de base : le rouge, le vert et le bleu. Or, nous disposons de $2^3 = 8$ couleurs sur le TO7. Lors de la formation du signal vidéo, il faut reconstituer ces couleurs à l'aide des bits contenus dans la mémoire d'écran. Pour que le décodage soit simple, il suffit de décider que le nombre binaire XYZ signifie, si X est à 1, que le bleu est allumé (et éteint dans le cas X = 0), et que de même Y représente le vert et Z le rouge. Ainsi sachant que les huit teintes sont obtenues de la façon suivante :

BLANC = ROUGE + VERT + BLEU
JAUNE = ROUGE + VERT
CYAN = VERT + BLEU
MAGENTA = ROUGE + BLEU
NOIR = aucune teinte
BLEU = BLEU
ROUGE = ROUGE
VERT = VERT

on a le codage :

Code	Couleur	bit du bleu	bit du vert	bit du rouge
0	NOIR	0	0	0
1	ROUGE	0	0	1
2	VERT	0	1	0
3	JAUNE	0	1	1
4	BLEU	1	0	0
5	MAGENTA	1	0	1
6	CYAN	1	1	0
7	BLANC	1	1	1

Sur le tableau ci-dessus, on constate que le nombre binaire constitué par le bit du bleu, le bit du vert et le bit du rouge (BVR) correspond au nombre décimal du code couleur. C'est pour cette raison que l'octet de couleur (deuxième plan mémoire) est codé en deux paquets de 3 bits.

En résumé

Pour allumer le secteur N à notre convenance, il faut :

1) Sélectionner le premier plan mémoire image en faisant

```
POKE &HE7C3, 1
     (ou l'équivalent assembleur)
```

2) Décider des points fond et graphisme en attribuant la valeur 1 aux bits correspondant aux points graphisme du secteur N et 0 aux points fond, coder le tout sur un octet que l'on placera à l'adresse &H4000 + N :

```
POKE &H4000 + N, &HXY
     (ou l'équivalent assembleur)
```

3) Sélectionner le deuxième plan mémoire image en faisant :

```
POKE &HE7C3, 0
     (ou l'équivalent assembleur)
```

4) Décider des couleurs fond et graphisme en mettant dans les bits 5 à 3 le code couleur de graphisme et dans les bits 2 à 0 le code couleur du fond, coder le tout sur un octet que l'on placera à l'adresse $4000 + N :

```
POKE &H4000 + N, &HX'Y'
     (ou l'équivalent assembleur)
```

4.5 LES INSTRUCTIONS GRAPHIQUES DU BASIC TO 7

4.5.1 Les instructions de commande de l'écran

Nous appelons instructions de commande de l'écran les instructions permettant de modifier une partie ou la totalité de l'image.

Dans la suite, les crochets [ ] entourant une expression ou un paramètre indiquent que l'expression ou le paramètre peut être omis(e).

* CONSOLE

Cette instruction fait appel à la notion de fenêtre de travail. La fenêtre de travail se délimite par une ligne supérieure et une ligne inférieure. Elle peut posséder une dimension verticale quelconque (entre 1 et 25 lignes de caractères) mais la largeur est toujours celle de l'écran, et en temps normal la fenêtre correspond à l'ensemble de l'image (lignes 0 à 24).

Cette fenêtre de travail est définie de façon à être utilisée par d'autres instructions telles que SCREEN et CLS ; ainsi, lors d'un effacement d'écran (RAZ ou CLS), seul le contenu de la fenêtre sera effacé.

La suite des manipulations décrites ci-dessous vous permettra de visualiser et de mieux comprendre l'intérêt de cette fenêtre.

En définissant une fenêtre de travail entre les lignes 5 et 15 à l'aide de

CONSOLE 5, 15 [ENTREE]

il est possible de changer uniquement la couleur de l'écran située entre les lignes 5 et 15. Pour cela, le fait de rentrer

SCREEN 2, 0 [ENTREE]

va colorer le fond de la fenêtre en noir et les caractères en vert. Si maintenant nous inscrivons dans cette fenêtre un message quelconque, le fait d'appuyer sur la touche RAZ effacera seulement le contenu de la fenêtre et le reste de l'écran demeurera inchangé.

Pour revenir à la configuration de départ, il faut tout d'abord restituer à la fenêtre de travail une hauteur de 25 lignes en entrant :

CONSOLE 0, 24 [ENTREE]

puis, pour le vérifier, appuyer sur la touche RAZ. On constate que la fenêtre occupe alors tout l'écran.

Un

SCREEN 4, 6 [ENTREE]

permet de revenir à la configuration initiale des couleurs.

Maintenant que la notion de fenêtre de travail est comprise, nous pouvons étudier l'instruction CONSOLE.

La syntaxe générale est :

CONSOLE [A] [, [B] [, [I] [, J]]]

A représente le numéro de la ligne supérieure de la fenêtre et N le numéro de la ligne inférieure. Bien entendu, A doit toujours être inférieur ou égal à B.

I est rarement utilisé, si I vaut 0 les caractères s'inscrivant avec la couleur qui leur a été spécifiée et si I vaut 1, la couleur ne changera pas bien que celle-ci reste enregistrée dans la mémoire d'écran et réapparaisse si l'on fait à nouveau I = 0.

J peut posséder 3 valeurs : 0, 1, ou 2.

Si J = 0, le défilement à l'intérieur de la fenêtre se fait à vitesse normale.

Si J = 1, le défilement se fait beaucoup plus lentement.

Si J = 2, lorsque la dernière ligne de la fenêtre est atteinte, le défilement reprend à partir de la première ligne.

Le programme suivant permet d'apprécier les différentes possibilités de défilement associées à la valeur de J.

```
  0  CONSOLE 5, 15, , 0:GOTO 100
  1  CONSOLE 5, 15, , 1:GOTO 100
  2  CONSOLE 5, 15, , 2:GOTO 100
100  LOCATE 0, 5 : CLS
110  FOR I = 1 TO 50
120  FOR J = 1 TO 20 : NEXT J
130  PRINT "I VAUT :" ; I
140  NEXT I : END
```

Après avoir rentré correctement le programme, faites tout d'abord un

RUN 0 [ENTREE]

Le texte défile du bas vers le haut relativement vite et dans la fenêtre de travail uniquement. Ceci correspond à J = 0 (ligne 0).

Maintenant faites un

RUN 1 [ENTREE]

(après avoir fait un RAZ). Un message identique au précédent défile mais avec une vitesse beaucoup moins rapide. Ceci correspond à J = 1.

Enfin, en effectuant un

RUN 2 [ENTREE]

vous pouvez observer un défilement du haut vers le bas et lorsque la dernière ligne de la fenêtre est atteinte, le défilement reprend à partir du haut.

Pour obtenir à nouveau une fenêtre normale exécuter la commande

CONSOLE 0, 24 [ENTREE]

* SCREEN

Cette instruction permet de changer les couleurs de la fenêtre de travail ainsi que du cadre. La syntaxe générale est :

SCREEN [A] [, [B] [, [C] [, [D]]]]

A indiqué le code de la couleur du graphisme (ou forme).

B indique le code de la couleur du fond.

C indique le code de la couleur du cadre.

(Le code des couleurs se trouve au paragraphe 2.4).

D indique lorsqu'il est présent que les couleurs fond et graphique doivent être interverties. D ne peut prendre que les valeurs 0 ou 1. En fait, on a l'équivalence des deux expressions

SCREEN A, B, C, D   et   SCREEN B, A, C

Les valeurs de A, B, et C sont comprises entre 0 et 7.

Exemples d'utilisation

La commande

```
SCREEN, , 1
```

permettra de modifier uniquement la couleur du cadre qui deviendra rouge.

La commande

```
SCREEN 0, , 3
```

changera la couleur des caractères qui deviendront noirs ainsi que celle du cadre qui deviendra jaune.

La commande

```
SCREEN 2, 0, 0
```

provoquera un changement de toutes les couleurs :

- les caractères deviendront verts ;
- le fond deviendra noir ;
- le cadre deviendra noir.

Définissons-nous maintenant une fenêtre de travail en exécutant

```
CONSOLE 7, 20 [ENTREE]
```

La commande

```
SCREEN 0, 3, 5 [ENTREE]
```

colore le cadre en mauve et à l'intérieur de la fenêtre de travail le fond sera jaune avec des caractères noirs. On constate que le reste de l'écran demeure inchangé : l'instruction SCREEN n'agit bien que sur la fenêtre de travail.

Grâce à la commande SCREEN l'utilisateur peut choisir la disposition des couleurs qu'il trouve la plus agréable et la plus reposante pour ses yeux. Ceci est important lorsqu'on passe de nombreuses heures devant un écran de téléviseur ; à ce sujet, nous recommandons les caractères verts sur fond noir obtenus par la commande

SCREEN 2, 0, 0 [ENTREE]

* CLS

Cette instruction permet d'effacer le contenu de la fenêtre de travail et de repositionner le curseur en haut à gauche dans cette même fenêtre. Aucun changement de couleur n'est effectué.

La syntaxe est CLS

* PSET

Cette instruction permet de positionner un point à un endroit quelconque de l'écran en mode graphique et une case quelconque en mode caractère.

- Mode graphique

La syntace est

PSET (X,Y) [, C]

X est la coordonnée horizontale du point à placer ($0 \leqslant X \leqslant 310$).

Y est la coordonnée verticale ($0 \leqslant Y \leqslant 199$) c'est-à-dire la ligne.

C doit avoir la valeur de la couleur désirée (voir paragraphe 2.4).

Si le point doit appartenir au graphisme alors : $0 \leqslant C \leqslant -1$.

Si le point doit appartenir au fond alors : $-8 \leqslant C \leqslant -1$.

Nous rappelons que deux points d'un même secteur ne peuvent posséder des couleurs de graphisme ou de fond différentes.

Exemples :

```
PSET (200,100), 1
```

met un point rouge graphisme en (200,100).

```
PSET (200,100), -5
```

met un point bleu en fond en (200, 100).

- Mode caractère

La syntaxe est :

PSET (X, Y) "Caractères" [, [A] [, [B] [, C]]]

X représente la colonne de la case à remplir (en mode caractères) donc $0 \leqslant X \leqslant 39$.

Y représente la ligne de la case à remplir ($0 \leqslant Y \leqslant 24$).

"Caractère" représente le caractère que l'on désire insérer dans la case de coordonnées (X,Y).

"Caractère" doit obligatoirement être une chaîne de caractères ou une expression chaîne. Seul le premier caractère de l'expression est pris en compte.

"Caractère" peut donc revêtir les formes suivantes :

. "E" ou "EXEMPLE" où E est un caractère du code ASCII.

. A$ qui représente une expression chaîne.

. CHR$(N) où N représente le code ASCII du caractère (il n'est donc pas possible d'utiliser les caractères semi-graphiques TELETEL).

. GR$(N) où N représente le numéro du caractère programmé par l'utilisateur.

A indique la couleur du caractère à afficher.

B indique la couleur du fond de la case (X,Y).

C indique lorsqu'il est présent que les couleurs A et B doivent être inversées donc :

PSET (X,Y) "C", A, B, C équivaut à PSET (X,Y) "C", B, A

C ne peut prèndre que les valeurs 0 ou 1.

Si A ou B sont omis, les valeurs par défaut seront les dernières valeurs utilisées. Il est d'ailleurs important de signaler qu'après l'exécution d'un programme, les caractères qu'affichera sur l'écran l'utilisateur auront la dernière couleur définie dans le programme si une instruction du type PSET, SCREEN, COLOR etc... a été utilisée. En conséquence, il est important avant de finir un programme de réinitialiser les couleurs.

Cependant, à la suite d'une mauvaise manoeuvre, il est possible de se trouver par exemple dans une configuration caractères noirs et fond noir. Il est alors impossible de vérifier que les caractères frappés au clavier l'ont été correctement. Pour échapper à cette situation, il convient d'appuyer sur la touche [ENTREE] de façon à être certain de ramener le curseur (invariable) en début de ligne (ou sur la touche RAZ pour le ramener en début d'écran) puis d'écrire au clavier (sans pouvoir le vérifier)

```
COLOR 4, 6 [ENTREE]
```

Exemples :

```
PSET (30, 10) "+" , 2, 1
```

va provoquer l'apparition d'un signe + de couleur verte sur fond rouge dans la case (30,10). Si l'instruction PSET (30, 20) "=" suit immédiatement, le signe = apparaîtra en (30, 20) avec les même couleurs que le signe +. Par contre, PSET (30, 20) "=" , , 6 aurait fait apparaître un signe = vert mais un fond cyan.

Lors de l'exécution du programme suivant :

```
10  A$ = ">"
20  SCREEN 6, 0, 0:CLS
30  PSET (5, 10) A$
40  PSET (10, 10) A$, 2, 1 : END
```

Le signe > sera affiché en (5, 10) avec une couleur cyan (ligne 30) puis en (10, 10) avec une couleur verte sur fond rouge. Après exécution du programme, les caractères que pourra écrire l'opérateur seront également verts sur fond rouge car aucune instruction modifiant ces dispositions n'existe avant la fin du programme (END).

L'instruction PSET (30, 10), CHR$(43), 2, 1 a exactement le même effet que PSET (30, 10) "+", 2, 1 puisque 43 est le code ASCII du signe +.

* DEFGR$

Cette instruction permet de définir les caractères de son choix et de les utiliser comme les caractères habituels du TO7.

La syntaxe est :

DEFGR$(N) = (liste d'entiers)

N représente le numéro de codage que vous attribuez au nouveau caractère ; N doit être compris entre 0 et 127.

(liste d'entiers) est une liste de 8 nombres entiers séparés par une virgule. Cette liste décrit le caractère (comme nous l'avons déjà expliqué au paragraphe 3.4) et est composée par conséquent de nombres compris entre 0 et 255. Comme tous les caractères du TO7, la dimension des caractères utilisateur est 8 X 8 soit 64 points.

La méthode que nous conseillons pour déterminer la liste d'entiers est la suivante :

- Dessiner le caractère désiré dans un carré composé de 64 petits carrés en ombrant les petits carrés (représentant les points) lorsqu'ils doivent appartenir à la couleur du graphisme (figure ci-dessous).

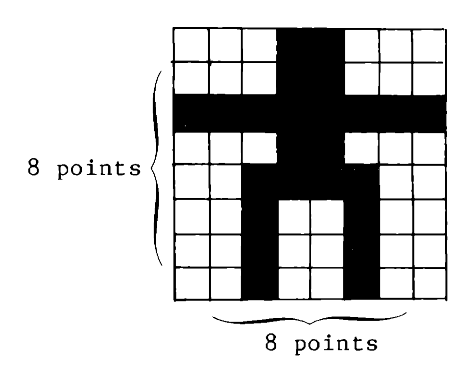


Figure 1

- Inscrire au dessus de chacune des 8 colonnes du grand carré et de gauche à droite les nombres suivants dans l'ordre : (figure 2)

128, 64, 32, 16, 8, 4, 2, 1

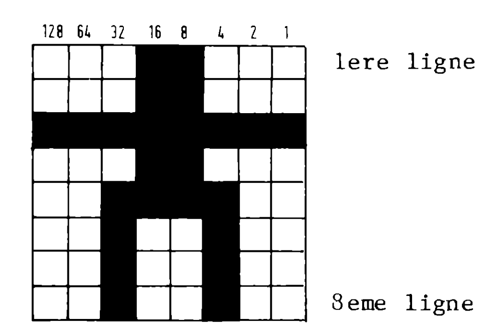


Figure 2

- Les 8 nombres se déterminent alors ligne par ligne et de haut en bas. Pour une ligne donnée ajouter entre eux tous les nombres se trouvant au-dessous des colonnes et dont les points sur cette ligne sont ombrés.

Par exemple, pour la première ligne, les points correspondant aux nombres 16 et 8 sont ombrés, donc le premier nombre de la liste sera 16 + 8 = 24 (figure 3).

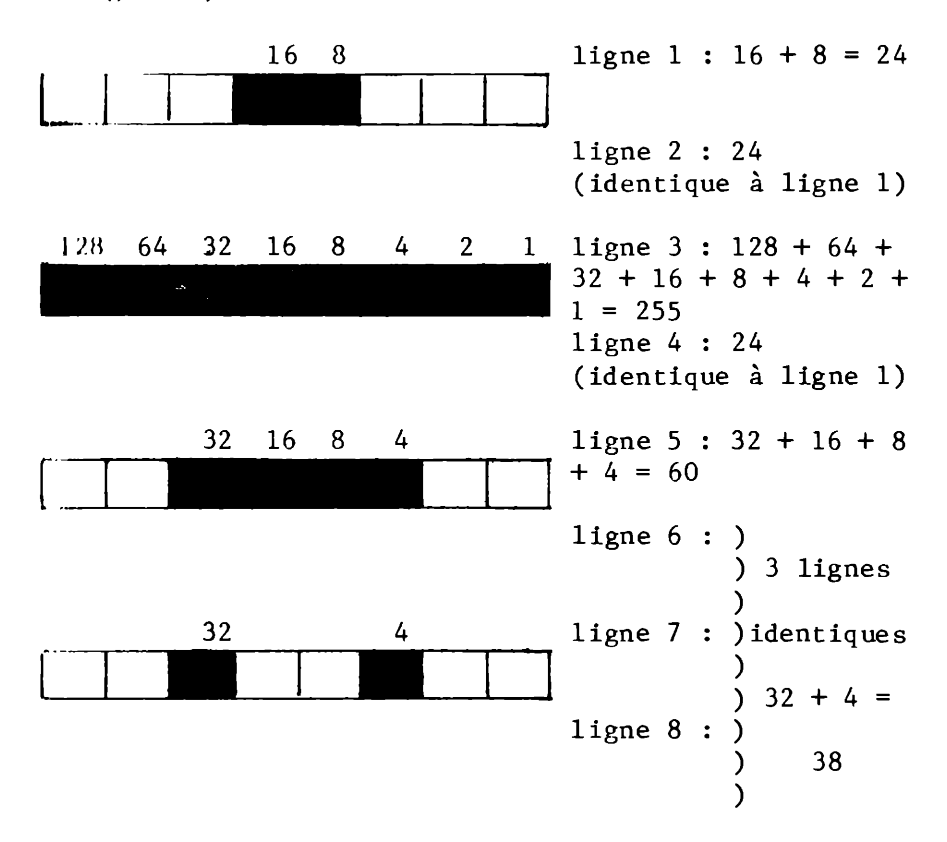


Figure 3

La liste d'entiers de l'exemple de la figure 1 est donc :

24, 24, 255, 24, 60, 38, 38, 38

L'instruction

DEFGR$(0) = 24, 24, 255, 24, 60, 38, 38, 38

définira sous le numéro 0 le caractère graphique de la figure 1.

Utilisation de DEFGR$

Un caractère pour pouvoir être défini par DEFGR$ doit au préalable se voir réserver une place mémoire à l'aide de l'instruction CLEAR. Nous rappelons que la syntaxe de l'instruction CLEAR était :

CLEAR [A] [, [B], C]]

C indique le nombre de caractères graphiques utilisateur à réserver pour la suite du programme.

Exemple :

Il vous faut définir 7 caractères utilisateur ; vous devez alors (de préférence en tout début de programme) utiliser l'instruction CLEAR ,, 7 avant de définir vos 7 caractères.

Les caractères une fois définis pourront être désignés soit par GR$(N) où N représente le code du caractère lors de sa déclaration par DEFGR$(N) : (liste d'entiers), soit par CHR$(128 + N).

Il est possible de créer jusqu'à 128 caractères utilisateur.

Exemples d'utilisation

Le programme suivant :

```
10  CLEAR ,, 1
20  DEFGR$(0) = 24, 24, 255, 24, 60, 38, 38,
    38
30  SCREEN 2, 0, 0 : CLS
40  PSET (10, 5) GR$(0)
50  PSET (11, 5) CHR$(128), 1
60  PRINT GR$(0)
70  COLOR 2 : END
```

réserve à la ligne 10 la place mémoire nécessaire à un caractère utilisateur, définit ce caractère à la ligne 20 (caractère de la figure 1) puis après avoir effacé l'écran (ligne 30) le dessine en 3 endroits.

A la ligne 40 on utilise l'instruction PSET et la désignation GR$(0), à la ligne 50 la désignation CHR$(128 + 0) = CHR$(128) puis un PRINT à la ligne 60. On constate que toutes ces syntaxes peuvent être utilisées de façons équivalentes.

* LINE

Cette instruction permet de tracer n'importe quelle droite entre deux points (X_1,Y_1) et (X_2, Y_2) de l'écran ; et ceci aussi bien en mode graphique qu'en mode caractère.

Mode graphique

La syntaxe est :

LINE $[(X_1, Y_1)]$ - (X_2, Y_2), C

X_1 et Y_1 sont les coordonnées du premier point $(0 \leqslant X_1 \leqslant 319$ et $0 \leqslant Y_1 \leqslant 199)$.

X_2 et Y_2 sont les coordonnées du deuxième point $(0 \leqslant X_2 \leqslant 319$ et $0 \leqslant Y_2 \leqslant 199)$.

C représente le code couleur de la ligne (voir paragraphe 2.4).

Si les points de la ligne doivent appartenir au graphisme alors $0 \leqslant C \leqslant 7$.

Si les points de la ligne doivent appartenir au fond alors $-8 \leqslant C \leqslant -1$.

Nous rappelons que deux points d'un même secteur ne peuvent posséder des couleurs de graphisme ou de fond différentes ; si une ligne passe dans le même secteur qu'un point ou qu'une autre ligne, il risque d'y avoir des modifications de couleurs indésirables.

Exemples :

LINE (100, 100) - (300, 150), 2

trace une ligne verte entre les points (100, 100) et (300, 150).

LINE - (300, 150), 2

trace une ligne verte entre les points (0, 0) et (300, 150).

Par contre, après un RAZ effectuez un

LINE (100, 100) - (200, 150), - 2 [ENTREE]

Contrairement à ce qui pourrait être imaginé en premier lieu, la ligne rouge est en fait devenue un escalier. Tout s'est passé comme si la couleur rouge avait déteint sur l'ensemble des secteurs coupés par la droite.

Ceci s'explique simplement par le fait que lorsqu'un point de la droite est inscrit dans un secteur, il y modifie la couleur du fond qui devient le rouge (-2), et par conséquent l'ensemble du secteur ayant à l'origine un fond bleu ciel aura désormais un fond rouge (si au départ vous étiez en configuration caractères bleu et fond cyan).

Le même type de problème se pose lors de l'intersection de deux droites. Après avoir effacé l'écran (RAZ), entrez successivement :

```
LINE (100, 100) - (200, 150), 1 [ENTREE]
LINE (100, 150) - (200, 100), 2 [ENTREE]
```

la première ligne trace une ligne rouge entre les points (100, 100) et (200, 150), la deuxième une ligne verte entre (100, 150) et (200, 100). Observez alors l'intersection des deux droites : une partie de la ligne rouge proche de l'intersection est devenue verte. L'explication est simple : lors du tracé de la ligne verte, les secteurs communs aux deux droites (proches de l'intersection) se sont vu attribuer une couleur de graphisme verte et les points graphiques initialement rouges sont restés des points graphiques mais ont pris la couleur verte.

Ces deux derniers exemples montrent les limitations annoncées au début de ce chapitre ; il conviendra donc à l'utilisateur de prendre des précautions particulières lorsque des points de couleurs différentes risquent de se trouver trop proches les uns des autres.

A ce stade, il peut être intéressant d'observer un phénomène très classique concernant le tracé de droites peu inclinées sur écran à balayage télévision. Traçons une droite rouge entre les points (0, 100) et (300, 120) :

```
LINE (0, 100) - (300, 120), 1 [ENTREE]
```

Nous constatons que la "droite" est en fait formée de paliers. Ceci ne constitue pas un défaut de votre ordinateur mais montre tout simplement une limitation liée aux écrans de télévision (l'algorithme de tracé de droites étant quant à lui tout à fait éprouvé).

Mode caractère

La syntaxe est :

LINE [(X_1, Y_1)] - (X_2, Y_2) "Caractère" [, [A] [, [B] [, C]]]

X_1 et Y_1 sont les coordonnées de la première case (début de la ligne) donc : $0 \leqslant X_1 \leqslant 39$ et $0 \leqslant Y_1 \leqslant 24$.

X_2 et Y_2 sont les coordonnées de la deuxième case (fin de la ligne), on a également: $0 \leqslant X_2 \leqslant 39$ et $0 \leqslant Y_2 \leqslant 24$.

"Caractère" représente le caractère qui s'inscrit dans toutes les cases par lesquelles passe la droite. "Caractère" doit obligatoirement être une chaîne ou une expression chaîne. Seul le premier caractère de l'expression sera pris en compte. "Caractère" peut revêtir les formes suivantes :

. "E" ou "EXEMPLE" où E est un caractère du code ASCII.

. A$ qui représente une expression chaîne.

. CHR$(N) où N représente le code ASCII du caractère (il n'est donc pas possible d'utiliser les caractères semi-graphiques TELETEL).

. GR$(N) où N représente le numéro du caractère programmé par l'utilisateur.

A indique la couleur du caractère à afficher.

B indique la couleur du fond des cases qui dessinent la ligne.

C indique par sa présence que les couleurs A et B doivent être interverties.

C ne peut prendre que les valeurs 0 et 1 (indifféremment).

Si A ou B sont omis, les valeurs par défaut seront les dernières valeurs utilisées. A ce stade, la même remarque pour l'instruction PSET en mode caractère est valable : après exécution du programme, les caractères entrés au clavier auront la dernière couleur définie durant le programme par ce type d'instruction, il convient donc d'être vigilant.

Exemples :

```
LINE (0, 10) - (30, 10) "+", 3, 0
```

va provoquer une ligne horizontale de 31 caractères + de couleur jaune sur fond noir.

```
LINE (20, 0) - (20, 20) CHR$(127), 5
```

trace une ligne verticale de couleur mauve. CHR$(127) est un caractère qui remplit toute la case (64 points en couleur graphisme).

```
LINE - (10, 20) "↑", 1, 0
```

provoque le tracé d'une ligne de caractères ↑ rouges sur fond blanc. Bien entendu, le défaut constaté en mode graphique est ici amplifié par la diminution de la définition.

Le programme ci-dessous montre un exemple de dessin simple à réaliser à l'aide des instructions SCREEN, PSET et LINE.

```
10  SCREEN 4, 0, 0 : CLS
20  LINE (0, 10) - (39, 10) CHR$(127)
30  LINE (0, 16) - (39, 16) CHR$(127)
40  FOR I = 0 TO 39 STEP 4
50  FOR J = 0 TO 1
60  PSET (I + J, 13) CHR$(127), 3
70  NEXT J : NEXT I
80  LOCATE 0, 0 : INPUT A$
90  SCREEN 4, 6, 6 : CLS : END
```

La ligne 10 impose le fond (noir) et indique la couleur graphique (bleu) qui va être utilisée aux lignes 20 et 30. Les lignes 20 et 30 tracent deux lignes parallèles de couleur bleue, cette couleur n'est pas indiquée puisque la valeur par défaut est 4 (définie à la ligne 10). Les lignes 40 à 70 tracent un pointillé jaune entre les deux lignes bleues. A la ligne 60, la couleur jaune (3) est indiquée puisque dans le cas contraire, la valeur par défaut aurait été le bleu. La ligne 80 permet de replacer le curseur en haut à gauche de l'écran en attendant que la touche [ENTREE] soit enfoncée ; la valeur A$ ne correspond à rien mais évite une erreur de syntaxe. Lorsque la touche [ENTREE] est enfoncée, le programme se termine non sans être retourné aux conditions initiales grâce à la ligne 90.

Ce dessin peut représenter une route pour un jeu vidéo ou pour une animation quelconque.

* BOX et BOXF

Ces deux instructions sont très utiles et permettent de tracer facilement des rectangles pleins ou "creux", elles présentent l'intérêt de remplacer plusieurs instructions LINE à la suite :

BOXF permet de tracer des rectangles pleins.

BOX ne trace que le périmètre du rectangle.

La syntaxe est très proche de celle de l'instruction LINE.

Mode graphique

La syntaxe est :

BOX[F] [(X_1, Y_1)] - (X_2, Y_2), C

Si F est absent (BOX), le rectangle sera tracé avec la même finesse que les lignes, c'est-à-dire avec une largeur de 1 point. Si F est présent (BOXF), le rectangle aura les mêmes dimensions que celui produit par BOX mais sera plein (toute la surface sera colorée).

(X_1, Y_1) représente les coordonnées du point le plus en haut à gauche du rectangle (figure 1).

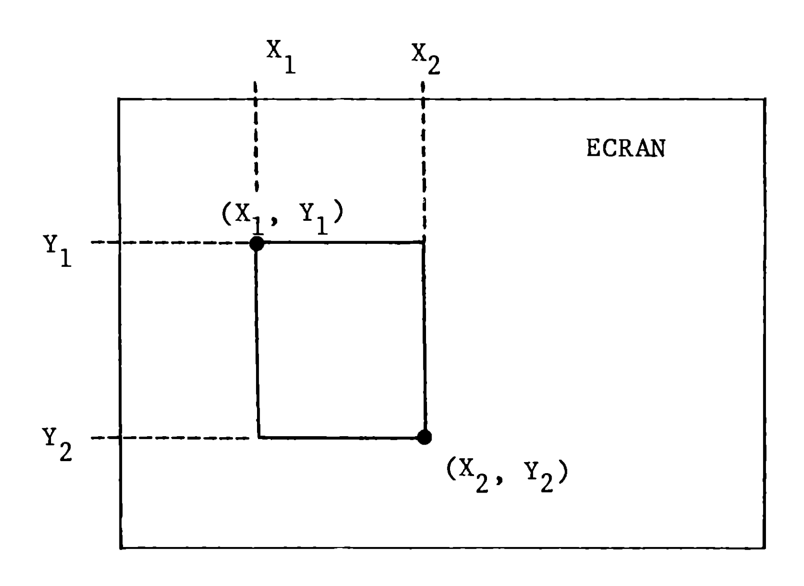
(X_2, Y_2) représente les coordonnées du point le plus en bas à droite du rectangle (figure 1).

En conséquence, les rectangles auront toujours des côtés parallèles aux bords de l'écran de télévision. D'autre part, si (X_1, Y_1) est omis, la valeur par défaut sera (0, 0).

C représente le code couleur du rectangle (voir paragraphe 2.4).

Si les points du rectangle doivent appartenir au graphisme, alors : $0 \leqslant C \leqslant 7$. Si les points doivent appartenir au fond alors : $-8 \leqslant C \leqslant -1$.

Nous rappelons que deux points d'un même secteur ne peuvent posséder des couleurs graphisme ou fond différentes ; si une partie du rectangle passe dans le même secteur qu'un point, une ligne ou une autre figure, il risque d'y avoir des modifications de couleur indésirables.



Exemple :

```
BOX (200, 10) - (280, 190), 1
```

trace un rectangle rouge ; seuls les côtés sont dessinés.

```
BOXF(200, 10) - (280, 190), 1
```

trace un rectangle rouge ; toute la surface du rectangle est colorée.

Mode caractère

La syntaxe est :

BOX[F] ([X_1, Y_1)] - (X_2, Y_2) "Caractère"
[, [A] [, [B] [, C]]]

Si F est écrit (BOXF), le rectangle verra toutes les cases par lesquelles passe sa surface contenir "Caractère". Si F est absent, seules les cases constituant le périmètre du rectangle contiendront "Caractère".

(X_1, Y_1) représente les coordonnées de la case la plus en haut à gauche du rectangle($0 \leqslant X_1 \leqslant 39$, $0 \leqslant Y_1 \leqslant 24$).

(X_2, Y_2) représente les coordonnées de la case la plus en bas à droite du rectangle.

"Caractère" représente le caractère qui s'inscrira dans les cases affectées par l'instruction BOX[F]. "Caractère" doit obligatoirement être une chaîne de caractères ou une expression chaîne. Seul le premier caractère de l'expression est pris en compte. "Caractère" peut revêtir les formes suivantes :

. "E" ou "EXEMPLE" où E est un caractère du code ASCII.

. A$ qui représente une expression chaîne.

. CHR$(N) où N représente le code ASCII du caractère (il n'est donc pas possible d'utiliser les caractères semi-graphiques TELETEL).

. GR$(N) où N représente le numéro du caractère graphique utilisateur.

A indique la couleur du caractère à afficher.

B indique la couleur du fond des cases qui dessinent le rectangle.

C indique par sa présence que les couleurs A et B doivent être inversées, C ne peut prendre que les valeurs 0 et 1 (indifféremment).

Si A ou B sont omis, les valeurs par défaut seront les dernières valeurs utilisées. Après exécution du programme, les caractères entrés au clavier auront la dernière couleur définie durant le programme par ce type d'instruction.

Exemples :

```
BOX (10, 10) - (30, 30) ">", 5, 0
```

trace un cadre de caractères > mauves sur fond noir.

```
BOXF (10, 10) - (30, 30) ">", 5, 0
```

trace un rectangle dont la surface est formée de caractères > mauves sur fond noir.

Le programme suivant :

```
10  SCREEN 0, 0 : CLS
20  BOXF (5, 10) - (14, 24) CHR$(127), 4
30  BOXF (15, 10) - (24, 24) CHR$(127), 7
40  BOXF (25, 10) - (34, 24) CHR$(127), 1
50  LOCATE 0, 0 : INPUT A$
60  SCREEN 4, 6, 6 : CLS : END
```

affiche un drapeau bleu, blanc, rouge sur fond noir. Les lignes 50 et 60 servent à revenir aux conditions initiales après exécution du programme.

* ATTRB et UNMASK

La première instruction permet d'une part de doubler la largeur et/ou la hauteur des caractères, et d'autre part de masquer provisoirement (ou non) des caractères. La deuxième complète ATTRB en démasquant tous les caractères masqués par ATTRB.

ATTRB

La syntaxe est :

ATTRB [L] [, [H] [, M]]

L vaut 0 pour une largeur normale et 1 pour une largeur double.

H vaut 0 pour une hauteur normale et 1 pour une hauteur doublée des caractères. Si la hauteur est doublée, le caractère occupe la ligne courante ainsi que la ligne précédente.

M vaut 0 si l'affichage se fait normalement et 1 si les caractères doivent être masqués. Le masque est noir dans tous les cas.

Dans le cas où un paramètre est absent, la valeur prise par défaut est celle définie par la dernière instruction de ce type ; à l'initialisation, les valeurs des trois paramètres L, H, et M sont 0.

UNMASK

Remarque : cette instruction n'agit que sur la fenêtre de travail ; on pourra donc utiliser l'instruction CONSOLE pour réaliser un démasquage sélectif.

Exemple 1 :

Le programme suivant montre différents formats d'affichage obtenus à l'aide de ATTRB.

```
 10  CLEAR, , 1 : SCREEN 1, 0, 0 : CLS
 20  DEFGR$(0)  =  177,  179,  183,  254,  254,
     183, 179, 177
 30  ATTRB 1, 1 : LOCATE 7, 2
 40  PRINT "DEMONSTRATION"
 50  COLOR 2 : PRINT GR$(0), "A"
 60  ATTRB,0
 70  PRINT GR$(0), "A"
 80  LOCATE 0, 9
 90  ATTRB 0, 1
100  PRINT GR$(0), " ", "A" LOCATE 0,23 : END
```

La ligne 30 définit un affichage double hauteur, double largeur. Le mot "démonstration" apparaît donc en grands caractères rouges sur l'écran. Ensuite sont affichés un caractère graphique et une lettre pour chacun des trois cas de démonstrations doublées. Le premier cas affiché est hauteur doublée, largeur doublée (ligne 50).

Le deuxième cas correspond à hauteur normale et largeur doublée (lignes 60 et 70), on remarque à la ligne 60 que L est omis puisqu'il a déjà été défini correctement à la ligne 30. Le dernier cas correspond à des caractères de largeur normale mais dont la hauteur est doublée (lignes 90 et 100).

Exemple 2 :

Cet exemple a pour but de mettre en évidence les possibilités de masquage et démasquage.

```
10  SCREEN 4, 6, 6 : CLS
20  PRINT "CODE SECRET :"
30  ATTRB,, 1
40  PRINT
50  A = INT (1000 * RND) : PRINT A
60  ATTRB,, 0 : LOCATE 0, 10
70  INPUT "AVEZ-VOUS TROUVE" ; REP$
80  UNMASK
```

Entre les lignes 30 et 60, tout ce qui est affiché à l'écran est masqué de noir. Lorsque l'on appuie sur la touche [ENTREE] (ligne 70) le masque est retiré (ligne 80). Si on remplace la ligne 10 par la ligne :

```
10  SCREEN 4, 6, 6 : CLS : CONSOLE 10, 24
```

alors le démasquage n'aura pas lieu étant donné que la partie masquée n'est plus dans la fenêtre de travail.

* COLOR

Cette instruction permet de changer la couleur graphisme et fond des prochains caractères et points affichés.

La syntaxe est :

COLOR (A) (,B) (,C))

A indique la nouvelle couleur des caractères.

B indique la nouvelle couleur de fond.

C indique par sa présence que les couleurs de A et B doivent être permutées. C ne peut prendre que les valeurs 0 et 1 (indifféremment).

Si A ou B sont omis, les valeurs par défaut seront les dernières valeurs utilisées par une instruction de ce type. Nous rappelons que A et B doivent avoir les valeurs suivantes :

0 pour le noir	5 pour le bleu
1 pour le rouge	6 pour le mauve
2 pour le vert	7 pour le cyan
3 pour le jaune	8 pour le blanc

Exemples :

```
COLOR 3, 0
```

provoque un affichage de tous les prochains caractères en jaune sur fond noir. Si cette instruction est suivie de :

```
COLOR , 1
```

alors les prochains caractères seront jaunes sur fond rouge, la valeur par défaut de A étant 3.

Cette instruction est très utilisée pour modifier la couleur des caractères que l'on frappe au clavier (notamment après un programme n'ayant pas réinitialisé les couleurs correctement).

Il faut également signaler que le fait d'appuyer sur la touche RAZ après une commande COLOR provoque une modification des couleurs de la fenêtre de travail qui après s'être effacée prend la couleur du fond définie lors de la commande A, B.

Exemple : si les caractères sont bleus sur fond cyan

COLOR 2, 0 [ENTREE]

provoque un changement de couleur pour tous les caractères entrés par la suite au clavier et une action sur la touche RAZ colore la fenêtre de travail en noir. On remarque que la couleur du cadre n'a pas changé.

4.5.2 Les fonctions de contrôle de l'écran

Nous appelons fonctions de contrôle de l'écran les fonctions permettant de déterminer l'état de l'écran ou d'une partie de l'écran à un moment donné. Ces fonctions permettent par exemple de trouver la position du curseur ou encore de savoir quelle est la couleur du point de coordonnées (X, Y).

Elles permettent également d'éviter la mise en mémoire systématique (ailleurs que dans la mémoire d'écran) d'informations concernant l'image créée.

* SCREEN

Cette fonction permet de déterminer le code ASCII du caractère se trouvant à l'intérieur d'une case quelconque de l'écran.

La syntaxe est :

SCREEN (X, Y)

X représente le numéro de la colonne où se situe la case.

Y représente le numéro de la ligne où se situe la case.

Nous sommes en mode graphique, on a $0 \leqslant X \leqslant 39$ et $0 \leqslant Y \leqslant 24$.

La valeur obtenue est le code ASCII (voir annexe) du caractère situé en (X, Y). Seuls les caractères du code ASCII sont reconnus. Les caractères graphiques utilisateur ne faisant pas partie du code ASCII ne seront pas identifiés et le code obtenu sera 0 comme tous les autres caractères non identifiables.

Le code des minuscules accentüées et du ç seront:

128	ç	136	è
129	a	137	î
130	â	138	ï
131	ä	139	ô
132	à	140	ö
133	é	141	û
134	ê	142	ü
135	ë	143	ù

Exemple :

Le programme suivant :

```
10  CLEAR ,, 1 : CLS
20  DEFGR$ (0) = 177, 179, 183, 254, 254,
    183, 179, 177
30  PSET (10, 10) GR$(0)
40  PSET (10, 15) "A"
50  LOCATE 0, 0
60  PRINT SCREEN (10, 10), SCREEN (10, 15)
```

inscrit tout d'abord un caractère graphique utilisateur en (10, 10) puis la lettre A en (10, 15). La ligne 60 affiche en haut de l'écran le code ASCII issu de l'instruction SCREEN de ces deux caractères. On obtient 0 pour le caractère utilisateur puisqu'il ne fait pas partie du code ASCII et 65 pour A.

* POINT

Cette fonction permet de déterminer la couleur de n'importe lequel des 64 000 points de l'écran.

La syntaxe est :

POINT (X, Y)

X représente la coordonnée horizontale du point ($0 \leqslant X \leqslant 319$).

Y représente la coordonnée verticale du point ($0 \leqslant Y \leqslant 199$).

On obtient une valeur comprise entre 0 et 7 si le point appartient au graphisme et entre -1 et -8 si le point appartient au fond. Le code des couleurs étant le code classique du mode graphique, nous avons :

pour le graphisme		pour le fond	
0	noir	-1	noir
1	rouge	-2	rouge
2	vert	-3	vert
3	jaune	-4	jaune
4	bleu	-5	bleu
5	magenta	-6	magenta
6	cyan	-7	cyan
7	blanc	-8	blanc

Exemple :

```
10  PSET (200, 100), 1
20  PRINT POINT (200, 100)
```

Ce programme inscrit un point rouge en (200, 100) puis imprime la valeur de la couleur du point (200, 100) qui est 2.

Utilisation de la fonction POINT en coordonnée "caractère"

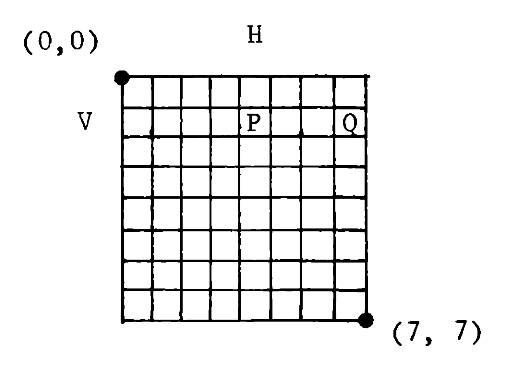
Dans beaucoup de jeux, ainsi que dans des programmes plus "sérieux", il est nécessaire de repérer un caractère graphique par sa couleur. Cette technique s'avère indispensable lorsqu'il s'agit des caractères graphiques utilisateur puisqu'il n'existe aucun autre moyen de les détecter.

Le problème est le suivant. Connaissant les coordonnées graphiques X et Y ($0 \leqslant X \leqslant 39$ et $0 \leqslant Y \leqslant 24$) d'une case, comment vérifier ou connaître la couleur du caractère s'y trouvant ?

Un raisonnement assez simple permet de déduire la formule suivante :

Code couleur = POINT (X * 8 + H, Y * 8 + V)

(X, Y) sont les coordonnées de la case scrutée, H et V sont les coordonnées du point scruté à l'intérieur de la case. Ces dernières coordonnées sont définies de la façon suivante ($0 \leqslant H \leqslant 7$ et $0 \leqslant V \leqslant 7$) :



Le point le plus en haut à gauche de la case a pour coordonnées (0, 0) et le plus en bas à droite (7, 7). Dans la figure ci-dessus, P a pour coordonnées (3, 3) et Q (5, 3). Par exemple, si la case a pour coordonnées X = 12 et Y = 16 et que l'on veut connaître la couleur du point Q (5, 3) dans cette case, il faudra utiliser la fonction

```
POINT (12 * 8 + 5, 16 * 8 + 3)
```

soit :

```
POINT (101, 131)
```

* CSRLIN

La syntaxe est :

```
CSRLIN
```

Cette fonction donne le numéro de la ligne où se trouve le curseur.

Exemple :

Le curseur se trouve sur la ligne 10 ; vous effectuez :

PRINT CSRLIN [ENTREE]

le résultat est 10.

* POS

La syntaxe est :

POS

Cette fonction donne le numéro de la colonne où se trouve le curseur.

Exemple :

Le curseur se trouve, après avoir écrit :

PRINT POS [ENTREE]

dans la colonne 0 puisque l'instruction PRINT fait passer à la ligne (la commande PRINT aussi). Le nombre affiché sera donc 0.

4.5.3 Entrées-sorties d'images

Nous avons regroupé dans ce paragraphe, les trois instructions permettant d'enregistrer ou de charger des images et (ou) de les imprimer.

* SAVEM

D'une façon générale, cette instruction permet de sauvegarder des programmes en binaire. La mémoire d'écran représentant un espace codé en binaire, il est également possible de la sauvegarder à l'aide de cette instruction

La syntaxe générale est :

SAVEM "[nom du programme]", [adresse D], [adresse F], [adresse exéc.]

"[nom du programme]" est du même type que celui employé par la commande SAVE.

[adresse D] est l'adresse de début du programme binaire ; pour l'image on a : [adresse D] = &H4000

[adresse F] est l'adresse de la fin du programme binaire ; pour l'image on a : [adresse F] = &H5F40

[adresse exéc.] est l'adresse de début d'exécution du programme ; pour l'image on choisira : [adresse exéc.] = &H4000

La mémoire d'écran étant composée de 2 plans mémoires situés à la même adresse mais sélectionnés par un buffer se trouvant à l'adresse &HE7C3 (voir paragraphe 4) le chargement couleur se fera en deux étapes.

Le programme-type de sauvegarde s'insèrera dans le programme dont on veut préserver l'image ; le listing sera le suivant :

```
XXXX      POKE &HE7C3, 1
XXXX+10   SAVEM "GRAPH", &H40000, &H5F40, 0
XXXX+20   POKE &HE7C3, 0
XXXX+30   SAVEM "GRAPH", &H40000, &H5F00, 0
XXXX+40   RETURN (dans le cas d'un sous-programme)
```

La ligne XXXX sélectionne le plan-mémoire graphisme, la ligne XXXX+10 le préserve sur bande magnétique. La ligne XXXX+20 sélectionne le plan-mémoire couleur et la ligne XXXX+30 le préserve sur bande magnétique.

* LOADM

Cette instruction complète la précédente en chargeant une image stockée sur bande magnétique.

La syntaxe est :

LOADM [" nom du programme "][,[décalage] [,R]]

En ce qui concerne l'image, décalage doit être nul. " nom du programme " a la même signification que pour l'instruction SAVEM.

Le programme type de chargement d'une image est :

```
XXXX      POKE &HE7C3,1
XXXX+10   LOADM "GRAPH"
XXXX+20   POKE &HE7C3,0
XXXX+30   LOADM "COUL"
XXXX+40   RETURN (dans le cas d'un sous-programme)
```

* SCREENPRINT

Cette instruction permet de recopier sur imprimante graphique le contenu de l'écran.

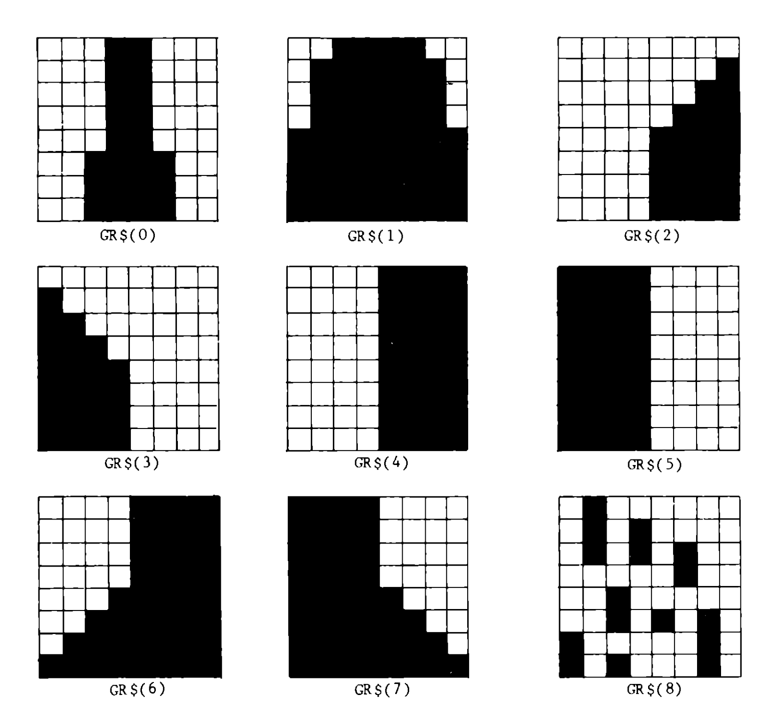
La syntaxe est :

SCREENPRINT

4.5.4 Exemples de programmes utilisant les instructions graphiques

Le programme pris en exemple et dont le listing se trouve ci-dessous a été conçu pour utiliser le plus d'instructions graphiques différentes possibles dans des configurations différentes. Le thème est l'animation du décollage d'une fusée.

Les caractères graphiques utilisateur sont définis aux lignes 20 à 100, les différents graphismes sont les suivants :



La fusée est dessinée à l'aide des 8 premiers caractères. Le dernier GR$(8) permet de dessiner la traînée de fusée (ligne 220).

Les lignes 200 à 230 assurent le déplacement de la fusée, les lignes 130 à 180 le compte à rebours.

Le sous-programme 1 000 dessine la figure à un emplacement dépendant de K dans le sens vertical (permet le déplacement). Le sous-programme 2000 dessine la tour de lancement.

La description détaillée de ce programme serait très longue. Cependant le lecteur soucieux d'apprendre et de se perfectionner aura tout intérêt (à titre d'exercice) à comprendre l'utilité de chaque ligne et de chaque instruction. Tous les éléments permettant de comprendre entièrement et de perfectionner ce programme ont été donnés.

```
 10  CLEAR,, 9 : SCREEN, 0, 0 : CLS
 20  DEFGR$(0) = 24, 24, 24, 24, 24, 60, 60,
     60
 30  DEFGR$(1) = 60, 126, 126, 126, 255, 255
     255, 255
 40  DEFGR$(2) = 0, 1, 3, 7, 15, 15, 15, 15
 50  DEFGR$(3) = 0, 128, 192, 224, 240, 240,
     240, 240
 60  DEFGR$(4) = 15, 15, 15, 15, 15, 15, 15,
     15
 70  DEFGR$(5) = 240, 240, 240, 240, 240,
     240, 240, 240
 80  DEFGR$(6) = 15, 15, 15, 15, 31, 63,
     127, 255
 90  DEFGR$(7) = 240, 240, 240, 240, 248,
     252, 254, 255
100  DEFGR$(8) = 64, 80, 84, 4, 32, 42, 130,
     162
110  GOSUB 2000
```

```
120  K = 0 : GOSUB 1000
130  ATTRB 1, 1
140  FOR I = 9 TO 0 STEP -1
150  LOCATE 0, 20 : PRINT I
160  FOR J = 1 TO 400 : NEXT J
170  NEXT I
180  ATTRB 0, 0
200  FOR K = 0 TO - 9 STEP -1
210  GOSUB 1000
220  LINE (18, 21 + K)-(20, 21 + K) GR$
     (8), 3  7
230  COLOR, 0 : NEXT K
240  COLOR 2 : END

1000  'FUSEE
1010  PSET (19, 16 + K) CHR$(127), 4
1020  PSET (19, 11 + K) GR$(0), 1
1030  PSET (19, 12 + K) GR$(1), 1
1040  FOR I = 0 TO 2
1050  PSET (19, 13 + I + K) CHR$(127)
1060  NEXT I
1070  PSET (18, 16 + K) GR$(2)
1080  PSET (20, 16 + K) GR$(3)
1090  FOR I = 0 TO 2
1100  PSET (18, 17 + I + K) GR$(4), 1
1110  PSET (19, 17 + I + K) CHR$(127), 2
1120  PSET (20, 17 + I + K) GR$(5), 1
1130  NEXT I
1140  PSET (18, 20 + K) GR$(6)
1150  PSET (20, 20 + K) GR$(7)
1160  PSET (19, 20 + K) CHR$(127)
1170  LINE (18, 21) - (20, 21) CHR$(127), 4
1180  RETURN

2000  FOR I = 0 TO 72 STEP 24
2010  LINE (176, 72 + I) - (192, 72 + I), 4
2020  LINE (176, 96 + I) - (192, 72 + I), 4
2030  LINE (176, 72 + I) - (192, 96 + I), 4
2040  NEXT I
```

```
2050  LINE (176, 72) - (176, 168), 4
2060  LINE (192, 72) - (192, 168), 4
2070  LINE (16, 21) - (24, 21) CHR$(127), 4
2080  RETURN
```

CHAPITRE **5**

Les sons et le TO 7

Ce chapitre a pour but d'étudier les possibilités sonores du TO7 et de montrer par quelques exemples comment mettre en oeuvre ces possibilités. Après avoir lu les quelques pages qui suivent, vous serez capable de composer de la musique sur votre TO7, de générer des effets sonores spéciaux pour agrémenter vos jeux : tir de mitraillette, sirène...

Un programme de jeu montre en particulier comment faire très rapidement un jeu simple mais attrayant grâce au graphisme et au son.

5.1 LE GÉNÉRATEUR DE SONS DU TO 7

Le TO7 est pourvu d'un générateur de sons très perfectionné qui lui permet de jouer de la musique sur une plage de cinq octaves. Le son est transmis par le haut-parleur du téléviseur.

Les sons produits par le TO7 sont les notes classiques (avec ou sans dièse et bémol). Le tempo, l'attaque et la durée de chaque note sont réglables.

5.2 LECTURE D'UNE CASSETTE

L'ensemble TO7 + lecteur-enregistreur de programmes peut se comporter comme un lecteur de cassettes normal.

Pour cela, il vous suffit de placer la cassette que vous désirez écouter dans le lecteur-enregistreur de programmes. Faites alors LOAD"" et pressez [ENTREE]. Pressez ensuite la touche PLAY du lecteur-enregistreur de programmes. Vous entendrez alors la musique enregistrée sur la cassette.

L'intérêt principal de cette possibilité n'est pas, bien sûr, d'écouter de la musique pré-enregistrée, la qualité de reproduction n'étant pas excellente... Ce procédé est particulièrement utile pour repérer les programmes enregistrés sur une cassette grâce à une annonce vocale (le nom du programme par exemple) ou musicale. L'annonce doit être enregistrée avec un magnétophone normal.

5.3 INSTRUCTION PLAY

L'instruction PLAY permet de faire "jouer" de la musique par le TO7.

La syntaxe de cette instruction est la suivante :

PLAY [expression chaîne], [expression chaîne]...

Les chaînes de caractères contenues dans expression chaîne doivent comporter un ou plusieurs éléments choisis obligatoirement parmi les cinq éléments suivants :

- note
- octave
- attaque
- tempo
- durée

Ces différents éléments peuvent être séparés les uns des autres par un point-virgule. Cela étant facultatif, le point-virgule sera systématiquement omis dans les exemples qui vont suivre.

Remarque : PLAY n'est pas seulement une instruction, c'est aussi une commande.

Faites

PLAY "DO RE MI FA SO LA SI DO" [ENTREE]

et vous entendrez le TO7 "monter" la gamme de do majeur.

5.3.1 Notes

Le TO7 peut jouer toutes les notes classiques : DO RE MI FA SOL LA SI.

Attention, la note SOL est codée par SO !!

Ces notes peuvent comporter un dièse, il suffit pour cela de faire suivre le nom de la note par le symbole # .

Exemple : DO#

Les notes peuvent aussi être avec bémol en faisant suivre leur nom par le symbole b (b minuscule).

Exemple : SI b

Le TO7 peut jouer aussi une note très particulière : le silence, noté P.

5.3.2 Octave

Le générateur de sons du TO7 a une plage de cinq octaves (de 1 à 5). O1 est l'octave la plus grave, O5 la plus aiguë.

PLAY "On" (avec n compris entre 1 et 5) sélectionne l'octave n. Tant qu'une autre instruction de même type n'aura pas été effectuée, toute note jouée sera à l'octave n.

Par défaut le TO7 joue à l'octave 4.

Exemples :

```
PLAY "DO RE MI RE DO"

PLAY "O2 DO RE MI RE DO"

PLAY "O5 DO RE MI O4 RE O3 DO"
```

5.3.3 Attaque

L'attaque de la note émise par le générateur de sons est réglable grâce à l'élément An où n est un entier compris entre 0 et 255.

A0 correspond à un son continu. Une valeur élevée pour l'attaque donne un son rapidement amorti. En pratique une attaque supérieure à 118 donne un son inaudible.

Le réglage de l'attaque est particulièrement intéressant pour obtenir des sons très secs. On peut ainsi, par exemple, simuler une rafale de mitraillette par :

PLAY "A110 T2 DO DO DO DO DO DO DO DO DO DO"

Ce genre de bruitage est fréquemment employé dans les programmes de jeux.

De même que pour le choix de l'octave, le choix d'une attaque donnée reste variable jusqu'à ce qu'un nouvel élément An soit rencontré.

Exemple :

PLAY "A2 DO DO A30 DO DO"

Par défaut l'attaque est A0.

5.3.4 Durée

Ln indique la durée relative des notes qui suivent cet élément, n étant un entier variant entre 1 (durée la plus courte) et 96 (durée la plus longue).

On peut poser comme convention : L24 correspond à une noire.

Cela permet d'établir le tableau suivant :

L 96=	ronde	L 72=	blanche pointée
L 48=	blanche	L 36=	noire pointée
L 24=	noire	L 18=	croche pointée
L 12=	croche	L 9=	double croche pointée
L 9 =	double croche		
L 3 =	triple croche		

Bien entendu toutes les durées intermédiaires sont possibles, ce qui permet d'obtenir des triolets, quintolets, etc...

Par défaut, le TO7 prend L 24.

Remarque : P (silence) est une note à part entière. Il est donc possible d'obtenir un soupir (L24 P), un demi-soupir (L12 P), etc...

Exemple :

```
PLAY "L6 DO RE L18 MI L6 SO L18 SO L6 LA L12
SO MI DO L6 DO RE L12 MI MI RE DO L24 RE L12
P"
```

5.3.5 Tempo

L'élément Tn (où n est un entier compris entre 1 et 255) règle la vitesse absolue de l'exécution en agissant dans les mêmes proportions sur les durées de toutes les notes qui suivent.

T1 est le tempo le plus rapide alors que T255 est le tempo le plus lent. Par défaut le tempo est pris à T5.

Exemple :

```
PLAY "T1 DO MI SO  MI DO T10 DO MI DO"
```

5.3.6 Exemples de mise en œuvre — Musique

* EXECUTION D'UN PETIT AIR DE MUSIQUE TRES CONNU

Entrez le programme suivant :

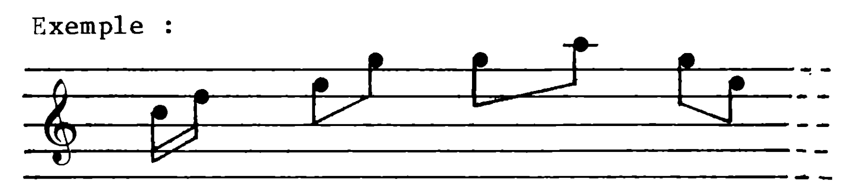
```
10  A$ = "T10 L6 DO RE L12 MI SO L18 SO 26 LA
    L12 SO MI DO L6 DO RE L12 MI MI RE"
20  B$ = "L12P L24 FA L36 FA L12 LA L36 LA
    L12 LA SO SO MI DO L24 RE L12 P"
30  C$ = "DO L24 RE L12P" : D$ = "L12 RE L24
    DO L12 P"
40  FOR I = 1 TO 2
50  PLAY A$ + C$ + A$ + D$ : PLAY B$ + A$ +
    D$
60  NEXT I
```

Vous reconnaîtrez certainement (du moins nous l'espérons) l'air : Oh Suzanna.

Ce programme appelle deux remarques :

- Il est préférable, pour programmer une musique, d'écrire d'abord cette musique sur une portée ; cela facilitera grandement la programmation :

Exemple :



L6 DO RE L12 MI SO L18 SO L6 LA L12 SO MI...

- On a tout intérêt à mettre dans des variables chaînes les séquences musicales qui reviennent plusieurs fois au cours d'un morceau. Le programme précédent en est un bon exemple.

Il pourra parfois s'avérer nécessaire de réserver de la place mémoire pour la manipulation des chaînes de caractères à l'aide de l'instruction CLEAR (reportez vous au chapitre sur le BASIC du TO7 pour des explications complètes de l'instruction CLEAR). Les chaînes "musicales" sont en effet souvent très longues et l'erreur OS peut être provoquée par le manque de mémoire nécessaire pour leur manipulation.

* — Sirène/alarme

Le fonctionnement d'une sirène est très simple : le son croît et décroît de façon continue et très rapidement.

Il est impossible de faire jouer au TO7 des sons espacés de moins d'un demi-ton. Cela n'est pas gênant dans la mesure où l'expérience montre qu'une gamme chromatique exécutée à grande vitesse produit quand même l'effet de sirène recherché. La vitesse d'exécution "gomme" la différence minimale d'un demi-ton entre les notes, et le son semble varier de façon continue.

Le programme ci-dessous génère quatre bruitages différents que vous pourrez utiliser avec profit dans vos programmes de jeux. Ils recouvrent en effet toute la gamme des bruits galactiques, rayons lasers, OVNI, etc... Le bruitage 4 est classiquement employé dans les jeux où l'on doit incrémenter ou décrémenter un score.

```
10  A$ = "DO DO♯ RE RE♯ MI FA SO SO♯ LA LA♯
    SI"
20  B$ = "SI LA♯ LA SO♯ SO FA♯ FA MI RE♯ RE
    DO♯ DO"
30  CLS : INPUT "1, 2, 3 ou 4 ?", N
40  N = INT (N) : IF (N - 1) * (N - 4) > 0
    THEN 30
50  ON N GOTO 100, 200, 300, 400
60  GOTO 30
100 FOR I  = 1 TO 10
110 PLAY "04 T1A0L6" + A$
120 NEXT I
130 RETURN
200 PLAY "TI A0L4"
210 FOR I = 1 TO 10
220 PLAY "04" + A$ + "05" + A$
230 NEXT I
240 RETURN
300 PLAY "TI A0L4"
310 FOR I = 1 TO 10
320 PLAY "03" + A$ + "04" + A$ + "05" + A$ +
    "04" + B$
330 NEXT I
340 RETURN
400 LOCATE 20, 15 : D = 0  : PRINT D : PLAY "
    L4 A0T1"
410 FOR I = 1 TO 10
420 PLAY "03" + A$ + "04" + A$ + "05" + A$ :
    D =  D + 1
430 LOCATE 20, 15 : PRINT D
440 NEXT I
450 FOR I = 1 TO 10
460 PLAY "05" + B$ + "04" + B$ + "03" + B$ :
    D =  D - 1
470 LOCATE 20, 15 : PRINT D
480 NEXT I
490 RETURN
```

* — Jeu : MISSILES

La principale application des sons concerne les jeux. Bien sonoriser un jeu est fondamental pour rendre celui-ci le plus attrayant possible. Les exemples précédents vous ont donné quelques aperçus de ce que l'on peut attendre du générateur de sons du TO7. Nous vous proposons maintenant un programme complet de jeu : MISSILES qui fait appel aux possibilités sonores du TO7.

Vingt cibles sont affichées en haut de l'écran. Un petit chariot mobile se déplace alternativement de droite à gauche et de gauche à droite dans le bas de l'écran. Quand vous pressez la touche F, le petit chariot s'arrête et tire un missile en direction des cibles. Vous disposez de trente missiles pour descendre le maximum possible de cibles.

```
   5  CLEAR, , 1 : GOSUB 10000
  10  FOR I  = 0 TO 39
  20  LOCATE I, 20 : PRINT "*"
  30  A$ = INKEY$ : IF A$ = "F" THEN GOSUB
      1000
  40  LOCATE I, 20 : PRINT " "
  50  NEXT I
  60  FOR I = 39 TO 0 STEP - 1
  70  LOCATE I, 20 : PRINT "*"
  80  A$ = INKEY$ : IF A$ = "F" THEN GOSUB
      1000
  90  LOCATE I, 20 : PRINT " "
 100  NEXT I
 110  GOTO 10
1000  PLAY "A0L4T104"
1010  B = B - 1
1020  COLOR 6, 0 : LOCATE 28, 22 : PRINT B
      : COLOR 2, 0
1030  IF B = 0 THEN END
1040  FOR J = 19 TO 1 STEP - 1
1050  LOCATE I, J : PRINT "+"
```

```
1060  PLAY A$ (J)
1070  LOCATE I, J : PRINT " "
1080  NEXT J
1090  IF POINT (4 + 8 * I, 4)  1 THEN
      RETURN
1100  LOCATE I, 0 : PRINT " "
1110  COLOR 6, 0
1120  FOR J = 1 TO 5 : S = S + 1 : LOCATE 7,
      22 : PRINT S : PLAY B$ + C$ : NEXT J
1130  COLOR 2,  0
1140  RETURN
10000 CLS : S = 0 : B = 30 : DIM A$ (26) :
      SCREEN 2, 0  0
10010 C$ = "DO DO# RE RE# MI FA FA# SO SO#
      LA LA# SI"
10020 B$ = "AOT1L4"
10030 FOR I = 1 TO 21
10040 READ A$(I)
10050 NEXT I
10060 DATA "SO", "FA#", "FA", "MI", "RE#",
      "RE", "DO#", "DO", "SO", "SI", "LA#",
      "LA", "SO#", "SO", "FA#", "FA", "MI",
      "RE#", "RE", "DO#", "DO"
10070 DEFGR$ (0) = 60, 126, 219, 255, 255,
      153, 153, 153
10080 ATTRB 1, 1 : COLOR 4, 0
10090 LOCATE 10, 12 : PRINT "MISSILES" :
      ATTRB 0, 0 : COLOR 2, 0
10100 LOCATE 0, 22 : INPUT "PRESSEZ ENTREE
      POUR COMMENCER", Q$ : CLS
10110 FOR I = 0 TO 38 STEP 2
10120 LOCATE I, 0, 0 : COLOR 1, 0 : PRINT
      GR$(0)
10130 NEXT I
10140 COLOR 5, 0
10150 LOCATE 0, 22 : PRINT "SCORE :" ; 5 :
      LOCATE 20, 22 : PRINT "BALLES :" ; B
10160 COLOR 2, 0
10170 RETURN
```

Nous espérons que ce jeu vous plaira et que vous y trouverez des idées pour faire vos propres jeux : bruitage du tir d'un missile, ...

Nous vous laissons le soin de sonoriser vous-même les jeux qui sont donnés dans le chapitre d'initiation au BASIC. Vous trouverez des versions élaborées de ces jeux ainsi que MISSILE dans le livre "JEUX SUR TO7" chez le même éditeur.

5.4 INSTRUCTION BEEP

Cette instruction, qui est aussi une commande, produit un son court identique à celui qui est produit à la frappe d'une touche de clavier.

Faire BEEP et PRINT CHR$(7) est équivalent car de façon standard le caractère de code 7 dans le code ASCII représente le bip sonore.

Exemple :

```
10  FOR I = 1 TO 10
20  R = INT (RND (1) * 10 + 1)
30  IF R = 1 THEN BEEP : END
40  NEXT I
50  PRINT CHR$(7)
60  END
```

CHAPITRE 6

Le crayon optique

6.1 INTRODUCTION

Le crayon optique permet de dessiner des figures variées directement sur l'écran du téléviseur. Son emploi peut être comparé à celui d'un crayon à papier sur une feuille, la feuille étant l'écran et le crayon à papier le crayon optique.

6.2 PRINCIPE GÉNÉRAL

Il peut paraître surprenant au premier abord qu'un crayon apparemment quelconque puisse autoriser une telle opération, la première question venant à l'esprit étant : "Comment le téléviseur peut-il savoir que le crayon se trouve là ?". La réponse est la suivante :

L'unité centrale du système TO7 génère elle-même l'image de télévision. Pour cela, il lui faut allumer un à un chaque point de l'écran de télévision dans un ordre rigoureux. L'ordre d'allumage est de haut en bas et de gauche à droite ; pour que l'oeil puisse percevoir ce phénomène, il est nécessaire d'éclairer chaque point 25 fois par seconde.

Un oeil suffisamment rapide observerait donc un à un les points de l'écran s'allumer et s'éteindre, il lui serait facile à un instant donné de vérifier qu'un point de son choix est noir ou coloré. Cet oeil existe, c'est l'élément photosensible du crayon lumineux que l'on peut observer au fond de son logement sous l'interrupteur. Le rôle de l'élément photosensible est d'avertir l'unité centrale lorsque le point situé à proximité du crayon optique est allumé. L'unité centrale en déduit alors facilement le numéro du secteur que pointe le crayon optique, il sera ensuite évident de changer la couleur du secteur correspondant par programme.

6.3 LES INSTRUCTIONS SPÉCIFIQUES AU CRAYON OPTIQUE

L'utilisation du crayon optique n'est possible que si des instructions permettent de déterminer sa position sur l'écran. Ces dernières sont décrites ci-dessous.

6.3.1 INPEN et INPUTPEN

Ces deux instructions permettent de relever la position du crayon optique sur l'écran sous forme de coordonnées graphiques.

La syntaxe est :

INPEN X, Y

ou INPUTPEN X, Y

La coordonnée horizontale est affectée à la variable X.

La coordonnée verticale est affectée à la variable Y.

Si l'opération se déroule normalement, on a :

$0 \leqslant X \leqslant 319$ et $0 \leqslant Y \leqslant 199$

Dans le cas contraire X et Y se voient attribuer la valeur -1. Ce dernier cas provient soit d'un manque de luminosité de l'image soit du fait que le crayon optique se trouve hors de l'écran.

INPEN X, Y provoque l'affectation immédiate des coordonnées aux variables X et Y alors que INPUTPEN X, Y n'effectue cette opération que lorsque l'interrupteur qui se trouve à l'extrémité du crayon est enfoncé.

Exemple 1 :

```
10  SCREEN 0, 7, 0 : CLS
20  INPEN A, B
30  CLS : PRINT A, B
40  FOR I = 1 TO 500 : NEXT I
50  GOTO 20
```

Ce programme colore la fenêtre de travail en blanc puis affiche à intervalles de temps réguliers les valeurs de A et B. En déplaçant le crayon lumineux sur l'écran, les valeurs A et B se modifient en fonction des coordonnées. Pour stopper le programme, presser simultanément les touches [CNT] et [C].

Exemple 2 :

```
10  SCREEN 0, 7, 0 : CLS
20  INPUTPEN X, Y
30  CLS : PRINT X, Y
40  GOTO 20
```

Le rôle de ce programme est identique au précédent.

Les zones devront préalablement être définies par une instruction PEN.

La syntaxe est

ONPEN GOTO N_0 [,[N_1][,[N_2] ...]]...]

ou ONPEN GOSUB N_0 [,[N_1][,[N_2] ...]]...]

N_0 à N_7 sont les numéros de ligne où doit se poursuivre le programme. N_0 correspond à la zone 0 et N_7 à la zone 7.

L'instruction n'est pas exécutée tant que l'interrupteur du crayon optique n'est pas enfoncé. Si aucune zone n'est pointée, l'instruction suivante est exécutée.

Exemple :

L'exemple suivant illustre l'utilisation des instructions PEN et ONPEN GOTO décrites précédemment.

```
 10  SCREEN  , 0, 0 : CLS
 20  PEN 0 ; (112,  136) - (136, 160)
 30  PEN 1 ; (176, 136) - (200, 160)
 40  PEN 2 ; (112, 0) - (200, 40)
 50  BOXF (112, 136) - (136, 160), 7
 60  BOXF (176, 136) - (200,160), 7
 70  BOXF (112, 0) - (200, 40), 7
 80  PSET (15, 18) "0", 4, 7
 90  PSET (23, 13) "1", 4, 7
100  LOCATE 18, 2 : COLOR 4, 7 : PRINT "END"
110  COLOR 2, 0 : ATTRB 1, 1
120  ONPEN GOTO 140, 150, 170
130  A = 2 : GOTO 160
140  A = 0 : GOTO 160
150  A = 1
160  LOCATE 16,11 : PRINT  A : GOTO 120
170  ATTRB 0, 0 : CLS : END
```

Les lignes 20 à 40 définissent trois zones numérotées 0, 1, 2 et les lignes 50 à 70 tracent trois rectangles blancs correspondant aux trois zones ; on remarquera l'étroite correspondance des coordonnées d'une zone et du rectangle associé (par exemple lignes 40 et 70). Les lignes 80 à 100 inscrivent du texte dans les trois zones. La ligne 120 est très importante ; si le crayon ne pointe aucune zone, la ligne 130 est exécutée ; par contre, si la zone 0, 1 ou 2 est détectée, la ligne 140, 150 ou 170 est exécutée (respectivement). Dans tous les cas, tant que l'interrupteur du crayon optique n'est pas enfoncé, le programme demeure en attente à la ligne 120.

Ce programme affiche le chiffre 0, si la zone 0 est pointée par le crayon ; 1 s'il s'agit de la zone 1 et 2 si aucune zone n'est pointée. Pour sortir du programme, la zone marquée END (zone 2) doit être visée par le crayon optique.

6.3.4 PTRIG

La syntaxe est :

PTRIG

Cette fonction indique l'état de l'interrupteur du crayon optique. Sa valeur est -1 lorsque l'interrupteur est enfoncé et 0 dans le cas contraire.

Exemple :

```
10  CLS
20  LOCATE 0, 0
30  PRINT PTRIG
40  GOTO 20
```

On constatera toutefois qu'il faut presser le crayon lumineux contre l'écran de télévision pour changer les valeurs de X et Y.

Ces deux exemples mettent en évidence le rôle des instructions INPEN et INPUTPEN ainsi que leurs différences.

Remarque : En fait, la coordonnée en X n'est pas donnée point par point mais secteur par secteur ainsi, la valeur la plus basse de X sera 3 (en dehors de -1) et X ne prendra que des valeurs de la forme 3 + H * 8 ($0 \leqslant H \leqslant 39$) ; H étant le numéro du secteur sur la ligne Y. .

Le programme suivant permet de dessiner sur l'écran à l'aide du crayon optique. Pour sortir du programme il suffit de presser la touche [S].

```
10  SCREEN 0, 7, 0 : CLS
20  IF INKEY$ = "S" THEN GOTO 70
30  INPEN U, V
40  IF (U < 0) or (V < 0) THEN GOTO 20
50  PSET (U,V), -5
60  GOTO 20
70  END
```

6.3.2 PEN

Cette instruction permet de définir sur l'écran des zones qui seront utilisées par les instructions ONPEN GOTO et ONPEN GOSUB. Il est possible de définir 8 zones rectangulaires.

* MODE GRAPHIQUE

La syntaxe est :

PEN N_1 ; (X1, Y1) - (X2, Y2) [, N_2 ; X'1, Y'1) - (X'2, Y'2)] [, ...

N_1, N_2, ... représentent des numéros de zone compris entre 0 et 7.

(X1, Y1) - (X2, Y2) représentent les coordonnées en mode graphique d'un rectangle tout comme l'instruction BOXF.

* MODE CARACTERE

La syntaxe est :

PEN N ; (X, Y)

N représent un numéro de zone et (X, Y) les coordonnées en mode caractère du carré définissant la zone. Ce mode est très restrictif puisqu'une zone ne peut occuper qu'une case de 64 points.

* MODE SUPPRESSION DE ZONE

La syntaxe est :

PEN [N;]

N représente le numéro de zone à supprimer ; si N est omis, toutes les zones sont supprimées.

Un exemple d'utilisation de cette instruction est donné à la fin du paragraphe suivant.

6.3.3 ONPEN GOTO et ONPEN GOSUB

Ces deux instructions permettent un branchement en fonction du numéro de zone pointé par le crayon optique.

Lorsque l'interrupteur est enfoncé le nombre -1 est affiché en haut à gauche de l'écran. Lorsqu'on le relâche, 0 apparaît.

6.4 CONCLUSION

Vous possédez maintenant les éléments nécessaires à l'utilisation du graphisme et du crayon optique. Il vous appartient d'exploiter au mieux vos connaissances pour créer des programmes qui n'auront que votre imagination pour limite.

Annexes

A. — Récapitulatif des instructions BASIC

* ATTRB X, Y, Z

Définit le mode d'affichage à l'écran : largeur double ou simple, hauteur double ou simple, mode masqué ou non.

* AUTO I, J

Permet de passer en numérotation automatique des numéros de ligne de programme (I, I+J, I+2J,...).

* BEEP

Produit une impulsion sonore.

* BOX (X1, Y1) - (X2, Y2), K

Dessine un rectangle de coins opposés (X1, Y1) et (XL, Y2) en couleur codée par K.

* BOXF

Idem BOX avec coloriage intérieur du rectangle.

* CLEAR

Permet de réserver de la place mémoire ou de créer des caractères graphiques etc...(cf chapitre 3).

* CLOSE I

Permet de fermer le canal I (et le fichier correspondant).

* CLS

Efface la totalité de l'écran.

* [CNT] [C]

Arrête l'exécution d'un programme.

* COLOR X, Y , Z

Permet de changer la couleur des caractères et du fond de l'écran.

* CONSOLE I, J, K, L

Définit la fenêtre de travail.

* CONT

Permet de reprendre l'exécution d'un programme.

* DATA

Permet d'introduire dans le programme une liste de données.

* DEFGR$

Permet de définir des caractères graphiques.

* DEFINT/DEFSNG/DEFDBL/DEFSTR

Permettent de déclarer le type des variables selon la première lettre de leur nom.

* DEF USR

Définit des sous-programmes écrits en langage machine.

* DELETE

Permet de détruire tout ou partie d'un programme (cf chapitre 3).

* DIM

Permet de dimensionner des matrices (ex : DIM A(20)).

* END

Indique la fin du programme principal.

* ERL, ERR

Variables contenant le numéro de la ligne ou s'est produite la dernière erreur, ainsi que le code de l'erreur.

* ERROR I

Permet de simuler l'erreur codée par le numéro I.

* EXEC adresse mémoire

Permet de lancer l'exécution d'un sous-programme écrit en langage machine.

* FOR ... TO ... STEP
  :
NEXT

Instruction de boucle (cf chapitre 3).

* GOSUB
  :
RETURN

Instruction de branchement vers un sous-programme (écrit en BASIC) (cf chapitre 3).

* GOTO

Instruction de branchement inconditionnel (cf chapitre 3).

* IF ... THEN ... ELSE ...

Instruction de contrôle permettant des branchements conditionnels (cf chapitre 3).

* INPEN, INPUTPEN

Permettent de lire les coordonnées d'un point de l'écran repéré par le crayon optique.

* INPUT

Permet d'introduire des données (au clavier) en cours d'exécution d'un programme (cf chapitre 3).

* INPUT#

Permet de lire des données inscrites sur un fichier.

* INPUTWAIT

Permet d'introduire des données en cours d'exécution d'un programme, tout en limitant le temps d'introduction.

* LET

Affecte une valeur à une variable.

* LINEINPUT

Permet d'introduire dans une variable chaîne de caractères une suite comportant plus de 255 caractères.

* LINEINPUT #

Permet de lire dans un fichier des suites de caractères et de les affecter à une variable chaîne de caractères.

* LIST

Permet de lister ou d'enregistrer tout ou partie d'un programme.

* LOAD

Permet de charger en mémoire centrale un programme.

* LOADM

Permet de charger en mémoire centrale un programme binaire.

* LOCATE I, J, K

Place le curseur à l'endroit désiré (coordonnées I, J) en le rendant visible ou non suivant la valeur de K.

* MID$

Permet des modifications de chaîne de caractères (cf chapitre 3).

* MOTORON

Permet de mettre le moteur du lecteur-enregistreur de cassette en marche.

* MOTOROFF

Provoque l'arrêt du moteur.

* MERGE

Permet de fusionner deux programmes.

* NEW

Efface le contenu de la zone mémoire reservée à l'utilisateur (programme et variables).

* ON ERROR GOTO

Permet de ne pas stopper l'exécution d'un programme si une erreur est rencontrée.

* ON ... GOTO
  ON ... GOSUB

Permettent des branchements conditionnels (cf chapitre 3).

* ONPEN GOTO
  ONPEN GOSUB

Permettent des branchements différents suivant la

zone de l'écran désignée par le crayon optique (cf chapitre 5).

* OPEN

Permet d'ouvrir un canal logique (et le fichier correspondant).

* PEN

Permet de définir des zones rectangulaires en leur attribuant des numéros.

* PLAY

Permet de jouer de la musique avec le TO7.

* POKE I, J

Place dans la case mémoire I la valeur représentée par J.

* PRINT

Permet l'affichage à l'écran de valeurs de variables ou de messages.

* PRINTUSING

Permet l'affichage à l'écran (comme PRINT), selon un format décrit par l'utilisateur, dans une expression chaîne appelée image (cf chapitre 3).

* READ A,...

Permet de lire des données inscrites dans un programme (cf instruction DATA) et de les affecter aux variables A etc...

* REM

Permet de placer des commentaires dans un programme.

* RESTORE

Permet de relire un jeu de données (placé après les instructions DATA).

* RESUME

Permet de revenir au programme principal après traitement d'un sous-programme d'erreur (cf chapitre 3).

* RUN

Lance l'exécution d'un programme : soit celui qui est situé en mémoire centrale, soit un programme situé sur un périphérique) (cf chapitre 3).

* SAVE [fichier]

Permet la sauvegarde d'un programme dans un fichier sur un périphérique donné.

* SAVEM

Enregistre une image-écran ou un programme écrit en langage machine.

* SCREEN I, J, K

Permet de modifier la couleur des caractères, du fond et du cadre (cf chapitre 4).

* SCREENPRINT

Recopie l'écran sur l'imprimante (référencée par PR90-040 exclusivement).

* SKIPF

Positionne la bande d'une cassette à un endroit voulu (cf annexe D).

* STOP

Stoppe l'exécution d'un programme.

* TRON

Permet de passer en mode trace (cf chapitre 3).

* TROFF

Permet de quitter le mode trace (cf chapitre 3).

* UNMASK

Permet de démasquer ce qui est à l'écran (qui a pu être masqué par la fonction ATTRB).

B. — Explication des messages d'erreur

Il existe 35 messages d'erreur différents. Ils sont explicités un par un dans l'ordre alphabétique de leurs symboles.

Remarque préliminaire : les messages d'erreur apparaissent de deux manières :

- en mode calcul le TO7 affiche un point d'interrogation, suivi du symbole correspondant à l'erreur détectée ainsi que du terme Error.

  Exemple :

  ? / 0 Error

- en mode programme, l'ordinateur affiche le même message (qu'il aurait imprimé en mode calcul) suivi du numéro de ligne où s'est produite l'erreur

  Exemple :

  ? / 0 Error In 100

  (l'erreur est apparu en (In) ligne 100)

* AO (code 52) : Already Open (déjà ouvert)

L'utilisateur a essayé d'ouvrir un canal déjà ouvert.

Exemple :

```
10 OPEN "I", 1, "ESSAI ONE"
20 OPEN "O", 1, "RESULT"
```

L'erreur AO est détectée en ligne 20, car le canal 1 se trouve déjà ouvert en ligne 10.

* BD (code 58) Bad Data (mauvaise donnée)

L'utilisateur a utilisé incorrectement les instructions READ et DATA. L'ordinateur a reçu l'ordre de lire une donnée. Le type de la donnée lue (numérique ou chaîne de caractères) n'est pas le même que le type de la donnée attendue.

Exemple :

```
 10  READ A$, NB
  :
100  DATA "JOSEPHINE", "SCORE"
```

A la lecture (ligne 10) le TO7 attend une chaîne de caractères et un réel et il se présente (ligne 100) deux chaînes de caractères.

Dans ce cas il faut vérifier les lignes "DATA et READ" et vérifiier si une instruction RESTORE (cf chapitre 3) n'a pas été oubliée dans le programme.

* BS (code 9) Subscript out of range (indice en dehors des limites)

Une matrice dimensionnée préalablement a été mal utilisée. L'ordinateur a reçu l'ordre de travailler avec une case inexistante de la matrice.

Exemple :

```
10  DIM A(5, 10)
20  INPUT A(5, 11), A(2, 8, 3)
```

En ligne 20 l'indice numéro 2, à savoir 11, est trop grand. Il ne doit pas dépasser 10. De plus la matrice A est de dimension 2 (ligne 10) donc il n'y a que deux indices.

Il faut vérifier :

- si les indices sont corrects ;
- quelles sont les instructions de DIMensionnement des matrices.

* CN (code 17) Can't CoNtinue

Cette erreur indique que l'exécution d'une suite d'instructions est impossible. Les raisons peuvent être multiples.

Exemple :

L'exécution d'un programme est interrompue par l'instruction [CNT] [C]. Si ce programme est modifié, une tentative de reprise d'exécution n'est pas possible.

* DD (code 10)

L'erreur apparaît uniquement lorsqu'un tableau (ou matrice) se trouve déclaré deux fois : implicitement ou explicitement.

Exemple :

```
10  DIM A(10)
20  DIM A(20)
```

* DS (code 56) Direct Statement

Cette erreur indique que l'utilisateur a tenté une commande directe dans un fichier (codé ASCII) qui se trouve en cours de chargement.

* DU (code 60) Device Unavailable (périphérique indisponible)

L'ordinateur a reçu l'ordre d'utiliser un périphérique mal raccordé ou inexistant. Il faut donc :

- renoncer à utiliser le périphérique,

ou

- vérifier si celui-ci est bien connecté au TO7.

* FC (code 5) Illegal Function Call (appel de fonction incorrect)

Cette erreur correspond à une mauvaise utilisation de fonction (au sens large). Elle peut se présenter dans de nombreuses circonstances :

Exemple 1 :

```
10  DIM A(5)
20  INPUT A(- 2) (l'indice de "localisation"
              est négatif, ce qui est incorrect)
```

Exemple 2 :

Avec POKE I, J si :
- . I est négatif ou supérieur à 65 535 ;
- . J est négatif ou supérieur à 255.

* FD (code 55) Bad File Descriptor (descripteur de fichier incorrect)

Le descripteur de fichier est incorrect.

Exemple 1 :

LIST "LPRT : (79100) ZONE"

Le nombre 79100 n'a aucune signification dans ce cas (cf descripteur de fichier chapitre 3).

Exemple 2 :

SAVE "COMM : (5880)"

Il manque le nom du fichier dans lequel doit être sauvegardé le programme (cf chapitre 3).

* FM (code 51) Bad File Mode (Mode d'accès au fichier incorrect)

Cette erreur indique que le mode d'accès au fichier est incorrect.

Exemple 1 :

```
OPEN "I", 4, "ZO"
PRINT#4, 890
```

L'ouverture du fichier s'est faite par le canal 4 en Input, soit dans le sens périphérique vers TO7. Or l'instruction PRINT est réservée aux communications TO7 vers périphérique.

Exemple 2 :

Avec la commande MERGE, le fichier à charger doit

être codé sous forme ASCII (sans quoi l'erreur FM est détectée).

* FN (code 23) For without Next (For sans Next)

Cette erreur correspond uniquement à une instruction de boucle mal définie. Il manque une instruction Next à la fin de la boucle (For ... Next).

Exemple :

```
10 DIM A(10, 10)
20 FOR I = 0 TO 10
30 FOR J = 0 TO 10
40 PRINT A(I, J)
50 NEXT J
```

Il faut écrire :

```
      50 NEXT J, I
ou   {50 NEXT J
     {60 NEXT I
```

* ID (code 12) Illegal Direct (commande incorrecte)

Cette erreur n'apparaît qu'en mode calcul. L'instruction utilisée est sans signification en mode calcul.

Exemple :

```
INPUT A [ENTREE]
```

* IE (code 54) Input past End

Cette erreur est relative à la lecture d'un fichier vide. En effet la lecture d'un fichier (à travers un canal) ne peut être effectuée que si celui-ci est non-vide.

Exemple :

```
10  OPEN "O", 3, "ZONE"   ) on ouvre le
                          ) fichier dans
                          ) lequel rien
20  CLOSE 3               ) n'est mis puis on
                          ) le referme

30  OPEN "I", 3, "ZONE"   ) on essaie de lire
40  INPUT#3, A            ) le fichier (vide)

50  CLOSE 3
```

* IO (code 53) Input Output (Entrée-sortie)

Elle est significative d'une erreur d'entrée-sortie, (Input, Output). Cela peut apparaître lors de l'utilisation d'un périphérique qui présente un défaut de fonctionnement.

* IU (code 59) Device In Use

Elle apparaît lors de l'utilisation d'un périphérique si ce dernier est déjà en utilisation.

* LS (code 15) String too Long (chaîne trop longue)

L'ordinateur se trouve avec une chaîne de caractères dépassant 255 caractères. L'utilisateur doit alors fractionner cette chaîne en un nombre suffisant de parties de sorte que chacune d'elles contiennent moins de 255 caractères.

* MO (code 22) Missing Operand (Opérande manquant)

Il manque un opérande dans une expression.

Exemple :

```
PRINT 2 +
```

Un nombre manque après l'addition.

* NF (code 1) Next without For (Next sans For)

Elle correspond à une instruction de boucle(s) mal définie. Deux cas se présentent :

1. Le TO7 a rencontré un NEXT se rapportant explicitement ou implicitement à une variable, sans avoir trouvé au préalable une instruction FOR relative à cette même variable (explicite ou implicite).

Exemple :

```
10  DIM A(10)
20  PRINT A(1)
30  NEXT        (où 30 NEXT I)
```

2. La suite des variables rencontrées dans la file des instructions FOR n'est pas retrouvée dans l'ordre inverse au fur et à mesure de la lecture des instructions NEXT. C'est le cas, par exemple, pour les boucles enchevêtrées.

Exemple :

```
10  FOR I = 0 TO 3
20  FOR J = 0 TO 3
30  INPUT A(I, J)
40  NEXT I, J
```

* NO (code 57) Not Open (non ouvert)

Le TO7 a reçu l'ordre de travailler avec un périphérique (sur un fichier) sans que le canal nécessaire à cette communication n'ait été ouvert. Il est donc nécessaire d'ouvrir un canal en Input ou Output selon le cas (cf chapitre 3).

Exemple :

```
10  INPUT#3, DONNEE, NOM$
```

Il faut ouvrir le canal 3 en Input.

*   NR (code 19) No Resume (pas d'instruction RESUME)

L'instruction RESUME est absente dans le programme. Elle est nécessaire dès qu'une instruction ON ERROR GOTO est placée dans un programme (cf chapitre 3).

Exemple :

```
10  ON ERROR GOTO 50
20  INPUT A , B
30  PRINT A/B
40  END
50  PRINT "B NE DOIT PAS ETRE NUL"
60  END
```

Si, à l'exécution, la valeur introduite pour B est 0 l'erreur NR sera révélée.

*   NU (code 50) Not in Use (pas en service)

Les canaux qui peuvent être ouverts et fermés sont au nombre de 16 (de 1 à 15). Cette erreur correspond à une tentative d'ouverture ou de fermeture d'un canal différent des 16 utilisables.

Exemple :

Open "O", 17, "DON" [ENTREE]

* OM (code 7) Out of Memory (dépassement de capacité)

Elle indique que la capacité de la zone mémoire réservée à l'utilisateur a été dépassée (il faut utiliser une extension mémoire, ou simplifier (alléger) si possible le programme ou les jeux de variables).

* OD (code 4) Out of Data (données épuisées)

Le TO7 peut recevoir l'ordre de lire des données, dans les lignes de programme repérées par l'instruction DATA, par l'intermédiaire d'une instruction READ. Si les données sont épuisées alors que le TO7 doit poursuivre la lecture, le message d'erreur OD est révélé.

Exemple :

```
 10  DIM  A(10)
 20  FOR I = 0 TO 10
 30  READ A(I)
 40  NEXT I
100  DATA 1, 3, 5, 7, 9
```

Il y a 5 données et 11 instructions de lecture (lignes 20 à 40).

* OS (code 14) Out of String Space

Le TO7 peut manipuler théoriquement des chaînes de caractères allant jusqu'à 255 caractères.

Il convient généralement de le prévenir afin qu'il réserve la place mémoire voulue (pour ces chaînes de caractères). Sans quoi, si par suite d'une ou de plusieurs concaténation(s) une "petite" chaîne grandit, l'ordinateur ne sera pas en mesure de la stocker (cf instruction CLEAR).

Exemple :

```
10  INPUT A$
20  FOR I = 1 TO INT (255/LEN(A$))
30  B$ = B$ + A$
40  NEXT
```

Remarque : Il est possible qu'avec le programme ci-dessus le message d'erreur n'apparaisse pas à tous les coups mais la probabilité d'apparition reste grande.

* OV (code 6) OVerflow

Une variable numérique a dépassé sa valeur limite.

Exemple :

```
A% = 32700   [ENTREE]
A% = 2 * A%  [ENTREE]
```

* PP (code 61) Protected Program (programme protégé)

Lorsque les programmes sont protégés, toute tentative

- de listage
- de modification du programme
- d'utilisation des fonctions PEEK, POKE etc...

se traduit par ce message d'erreur.

*   RE (code 20) Resume without Error (Resume sans Erreur)

Si, dans un programme, l'ordinateur rencontre une instruction RESUME sans avoir préalablement détecté une erreur, il s'arrête et affiche l'erreur RE.

Exemple :

```
10  ON ERROR GOTO 40
20  INPUT "DONNEZ DEUX NOMBRES", A, B
30  PRINT A/B
40  RESUME
```

Si, à l'exécution, la valeur introduite par B est différente de 0, l'erreur RE apparaît.

Remarque : en pratique et généralement, les instructions RESUME ne doivent pas faire partie du programme principal mais plutôt du sous-programme (ou du moins être situées après le END final).

Exemple :

```
10  ON ERROR GOTO 50
20  INPUT "DONNEZ DEUX NOMBRES", A, B
30  PRINT A/B
40  END
50  RESUME
```

*   RG (code 3) Return without Gosub (Retour sans Gosub)

Dans un programme le TO7 a rencontré une instruction RETURN sans avoir préalablement trouvé une instruction GOSUB. Cette erreur est propre aux sous-

programmes insérés dans le programme principal au lieu d'être placés à sa suite (après le END final).

Exemple :

```
10  PRINT "TAPEZ VOTRE MISE (POSITIVE)"
20  INPUT "MISE", MISE
30  IF MISE  = 0 THEN GOSUB 10
 :
```

Remarque : Il faut utiliser un GOTO à la place du GOSUB ou déplacer le sous-programme après le END final.

*   SN (code 2) Syntax Error (Erreur de syntaxe)

Elle indique une erreur de syntaxe. Cette erreur est relativement courante. Il faut vérifier la ponctuation, les espacements, les parenthèses, les guillemets, etc...

Exemple :

```
10  INPUT "VOTRE NOM S.V.P.", NOM$
20  PRINT "BONJOUR", NOM$
```

*   ST (code 16) String formula Too complex

Une expression sur chaîne trop complexe détermine cette erreur.

*   TM (code 13) Type Mismatch

Cette erreur est relative à un conflit de type. L'ordinateur reçoit l'ordre de mettre une chaîne de caractères dans une mémoire affectée à des valeurs numériques ou l'inverse.

Exemple :

```
10  INPUT "NOM", A$
20  B = A$ + A$
```

* UE (code 21) Undefined Error (erreur non définie)

L'instruction ERROR (cf chapitre 3) a été utilisée avec un code d'erreur incorrect.

Exemple :

```
10  ERROR 80      (l'erreur 80 n'existe pas)
```

* UF (code 18) Undefined User Fonction

Le TO7 reçoit l'ordre d'utiliser une fonction non définie. Ceci se produit avec les fonctions USR (cf chapitre 3).

Exemple :

```
10  Y = USR 11 (X)
```

Les fonctions USR n'ont que 10 numéros possible de 0 à 9, donc 11 n'est pas accepté.

* UL (code 8) Undefined Line

Cette erreur apparaît avec l'utilisation des instructions de branchement (GOTO, GOSUB, etc...), lorsque la (ou les) ligne(s) de branchement n'existe(nt) pas.

Exemple :

```
 50  GOTO 99
  :
 90  PRINT SCORE
100  INPUT "VOULEZ VOUS REJOUER", A$
  :
```

*   /0 (code 11)

Cette erreur correspond à la division par zéro.

Exemple :

```
PRINT A/0 [ENTREE]
```

*   Notons l'erreur : ? Redo qui joue un rôle un peu à part.

Cette erreur est révélée lorsque le TO7 attend une donnée de la part de l'utilisateur (cf instruction INPUT) et que ce dernier opère une mauvaise Introduction. Il faut alors réintroduire correctement (si possible) la donnée.

Exemple :

MISE ?	3, 14 [ENTREE]
cette partie est affichée par le TO7 grâce au proramme suivant : 10 INPUT "MISE", A	cette partie est tapée par l'utilisateur (elle est incorrecte car la virgule doit être remplacée par un point).

C. — Utilisation d'une cassette

1. GENERALITES

1.1 Le lecteur-enregistreur a deux fonctions :

- enregistrer sur la cassette l'information en provenance de l'ordinateur ;

- lire sur la cassette l'information à destination du TO7.

L'information peut être de plusieurs types :

- un programme écrit en BASIC,

- un programme écrit en langage machine,

- un ensemble de données.

1.2 Raccordement du lecteur-enregistreur

Il se fait au moyen de deux cables à savoir :

- l'alimentation (du lecteur avec le secteur 220 V alternatif),

- la liaison avec le TO7 (le câble est terminé par une prise DIN qui est à brancher sur le flanc gauche du TO7).

Si une cassette* est introduite et si tout est mis sous tension (TO7 + lecteur) la manoeuvre de :

- la touche [▷▷] ou [◁◁] fait défiler la bande selon le sens indiqué ;

- les touches [▷] ou [▷] et [enregistrement] sont sans effet apparent sauf s'il y a un ordre provenant du TO7 (cet ordre est tapé par l'utilisateur).

Remarque : les syntaxes qui se trouvent dans cette annexe concernent exclusivement l'utilisation de la cassette.

2. INSTRUCTIONS D'ENREGISTREMENT (sur cassette)

Ces instructions supposent que le lecteur-enregistreur se trouve connecté et que les touches lecture et enregistrement ont été enfoncées (le voyant rouge doit se trouver allumé).

2.1 Enregistrement de programme ou de partie de programmme

2.1.1 SAVE permet d'enregistrer sous l'une des trois formes suivantes la totalité du programme BASIC situé dans la mémoire centrale (du TO7) à savoir :

- sous forme conventionnelle ;

- sous forme codée ;

- sous forme protégée.

(*) En principe toute cassette audio peut convenir, mais une cassette de bonne qualité est cependant recommandée.

Exemples de chacune des trois formes :

- 1 SAVE "JEU"

- 2 SAVE "JEU", A (A pour ASCII)

- 3 SAVE "JEU", P (P pour protégé)

Remarque : dans les trois cas la cassette défile un certain temps puis s'arrête et le message OK, signalant la fin de l'enregistrement, s'affiche à l'écran.

2.1.2 LIST

Syntaxe :

LIST "CASS : nom de fichier " [, plage de numéro de ligne]

Permet d'enregistrer, sous forme codée ASCII, un ensemble de lignes de programme (indiqué par plage de numéro de ligne). Il est nécessaire de préciser le nom du périphérique soit CASS sans quoi le TO7 considère d'autorité l'imprimante.

Remarques :

. si la plage de numéro de ligne est omise dans la commande, cette instruction produira le même effet que :

SAVE " nom de fichier ", A

(la totalité du programme est enregistrée)

. Voir la remarque commande SAVE ci-avant.

Exemple :

LIST "CASS : JEU", 50 - [ENTREE]

enregistre les lignes 50 et suivantes.

2.2 Enregistrement, dans des fichiers binaires, de programmes écrits en langage machine ou d'images écran

Remarque : ce type de fichier sert très peu ou pas du tout lors d'une première approche.

Les adresses mémoires où se trouve le programme dans l'unité centrale doivent être connues ainsi que l'adresse de lancemenent de ce programme.

Les images écran sont stockées dans les mémoires situées de &H4000 à &H5F40 (cf chapitre 4).

Moyennant la connaissance des adresses (mémoires) l'instruction SAVEM (syntaxe SAVEM adresse de début de programme , adresse de fin de programme , adresse d'exécution) provoque l'enregistrement sur cassette du contenu des mémoires ainsi désignées.

2.3 Enregistrement de données (dans un fichier de données)

Il se fait en trois temps :

a) Ouverture d'un canal en Output et par là-même d'un fichier permettant le dialogue TO7 vers lecteur-enregistreur (cf instruction Open chapitre 3).

Exemple :

10 OPEN "O", #1, "DONNEES"

La bande de la cassette se met à défiler durant quelques secondes et s'arrête.

Le canal 1 est ouvert en Output vers le fichier DONNEES situé sur cassette.

b) Envoi des données à enregistrer

(cf instruction PRINT et PRINT USING chapitre 3).

Exemple :

Reprendre l'exemple précédent en ajoutant les lignes suivantes :

```
20  INPUT "VOULEZ-VOUS RENTRER UNE DONNEE ?
    OUI - NON" ; REPONSE$
30  IF REPONSE$ = "NON" THEN GOTO 70
40  INPUT "TAPEZ LA" ; A$
50  PRINT#1, A$
60  GOTO 20
70  END
```

Remarques :

. Si on désire entrer des valeurs numériques il est nécessaire de remplacer A$ en ligne 40 et 50 ci-dessus par A ou A#.

. Dans cette deuxième phase la cassette peut ne pas avancer mais s'il y a défilement il se fait par saccades (ceci est dû à la présence de mémoire tampon ce qui ne fait pas l'objet de ce developpement).

c) Fermeture du canal et du fichier

Cette phase est importante car elle sert à repérer sur la bande de la cassette la fin du fichier.

Exemple :

```
70 CLOSE 1
80 END
```

Dans cette phase la cassette défile toujours durant quelques instants.

3. INSTRUCTIONS DE LECTURE (sur cassette)

Le lecteur-enregistreur doit être connecté et la touche lecture [▶] doit être préalablement enfoncée.

3.1 Lecture d'un programme

3.1.1 LOAD

Syntaxe :

LOAD " nom de fichier "

Permet de charger en mémoire centrale la totalité du programme enregistré dans la cassette sous le nom (nom de fichier) donné.

Exemple :

LOAD "JEUX 3" [ENTREE]

La bande se met à défiler, le message Searching (en recherche) apparaît sur l'écran, puis le TO7 affiche le nom des fichiers rencontrés soit par exemple :

Searching		ces termes indiquent si
Skip JEUX 1	BAS	le fichier est BASIC
Searching		(BAS) ou binaire (BIN)
Skip DONNEES	DAT	ou de données (DAT)
Searching		

Jusqu'à trouver le fichier demandé (JEUX 3) et alors le message suivant s'affiche :

FOUND JEUX 3 BAS

La bande continue de défiler (le TO7 lit ce qu'il y a dans le fichier). Après un temps plus ou moins long suivant la taille du fichier, le défilement est stoppé et le TO7 affiche

OK

si aucune erreur n'a été rencontrée, sinon le message d'erreur correspondant s'affiche.

Remarques :

. si le nom de fichier est omis le TO7 charge le premier rencontré ;

. si le fichier indiqué n'a pas été trouvé avant la fin de la cassette, il faut réinitialiser le TO7.

3.1.2 Commande MERGE

Syntaxe :

MERGE " nom du fichier "[,R]

Cette instruction permet :

- de charger le programme situé dans la cassette sous le nom nom de fichier (exemple sous le nom JEUX 3),

<u>et</u> - de le fusionner au programme éventuellement présent dans le TO7 (cf chapitre 3).

Cette commande est peu utilisée par les néophytes.

Exemple :

```
LOAD "PGM1"  [ENTREE]  (le fichier PGM1 est
                        chargé)
MERGE "PGM2" [ENTREE]  (le fichier PGM2 va
                        être fusionné avec
                        PGM1)
```

Remarques :

. le caractère R (qu'il est possible de rajouter à la suite de la commande MERGE) ordonne au TO7 de lancer l'exécution du programme sitôt la fusion effectuée. De ce fait,

```
MERGE " nom de fichier ", R [ENTREE]
```

est équivalente à :

```
MERGE " nom de fichier "  [ENTREE]
RUN [ENTREE]
```

. si le nom du fichier est omis le TO7 prend en compte le premier fichier BASIC rencontré.

3.2 Lecture d'un fichier binaire contenant un programme écrit en langage machine ou d'une image écran

Il faut connaître l'adresse des mémoires dans lesquelles va être placé le contenu du fichier ainsi que les adresses mémoire dans lesquelles le fichier était placé lors de son enregistrement sur cassette. Si ces adresses mémoire ne correspondent pas il y a un décalage d'adressage. Moyennant la connaissance de la valeur de ce décalage la lecture peut être effectuée par l'instruction LOADM (cf chapitre 3).

Exemples :

1. LOADM "FIGURE"   [ENTREE]
2. LOADM "FONCT1", 50 [ENTREE]

50 représente le décalage d'adressage.

La cassette doit défiler jusqu'à trouver le fichier correspondant puis s'arrêter après le chargement (opéré).

3.3 Lecture de fichier de données

Elle se fait en trois temps :

a) Ouverture d'un canal en <u>Input</u>

Exemple :

```
10  OPEN "I", # 12, "DONNEES"
```

La bande défile - à l'exécution - jusqu'à trouver le fichier DONNEES après quoi elle s'arrête et il est alors seulement possible de passer à la suite des opérations.

b) Lecture des données inscrites sur le fichier

Cf instructions INPUT #, INPUT$, LINEINPUT # chapitre 3).

Exemple :

Rajouter à la ligne 10 de l'exemple précédent :

```
20  IF EOF (12) THEN GOTO 60
30  INPUT # 12, A
40  PRINT A
50  GOTO 20
60  END
```

Remarque : ainsi que pour PRINT# la bande ne défile pas continûment.

c) Fermeture du fichier et du canal ouvert

Cette opération est indispensable pour que le TO7 puisse travailler, par la suite, sur d'autres fichiers (sur cette même cassette) sinon un message d'erreur peut apparaître.

Exemple :

Remplacer dans l'exemple précédent (cf ci-dessus) la ligne 60 par :

60 CLOSE 12

4. INSTRUCTIONS DIVERSES

En plus des instructions examinées dans les paragraphes précédents il en existe trois autres.

4.1 SKIPF

Syntaxe SKIPF " nom de fichier ")

Elle permet de positionner la bande de la cassette juste après le fichier indiqué par nom de fichier. Si ce dernier ne figure pas sur la cassette, la bande défile jusqu'au bout et le TO7 doit être réinitialisé. Cette instruction n'est guère utilisée si l'utilisateur procède avec méthode lors de l'emploi des cassettes (cf paragraphe 4 ci-après).

Exemple :

SKIPF "JEUX 3" [ENTREE]

Remarque : si le nom du fichier n'est pas précisé le TO7 place la bande de la cassette après le premier fichier rencontré.

4.2 MOTORON - MOTOROFF

Ces instructions permettent respectivement de mettre en route et d'arrêter le défilement de la bande. Elles sont utilisées pour effacer des fichiers ou des cassettes chargées de fichiers. Pour ce faire il faut procéder comme suit :

a) Positionner la bande de la cassette (avec par exemple les indications de position affichées par le compteur) au moyen des touches [▶▶] ou [▷▷] juste avant la zone à effacer.

b) Enfoncer simultanément les touches lecture [▶] et [enregistrement]. Le voyant rouge s'allume.

c) Taper l'instruction suivante sur le TO7 :

MOTORON [ENTREE]

qui provoque l'effacement du contenu de la cassette.

d) L'arrêt du défilement s'obtient en tapant :

MOTOROFF [ENTREE]

5. CONSEILS PRATIQUES POUR L'UTILISATION DE LA CASSETTE

* Il est important de bien inscrire sur chaque cassette :

- le nom de chacun des fichiers enregistrés,

- le type de leur contenu (fichier programme BASIC, fichier binaire, fichier données),

et - le repère de début et, éventuellement, de fin de fichier, indiqué par le compteur du lecteur-enregistreur.

* D'autre part il est utile de réserver un espace pour séparer entre chaque fichier enregistré. Cet espace doit correspondre à un défilement minimum de 5 unités de compteur.

* A la lecture d'un fichier donné, il est préférable de faire défiler la cassette au moyen des touches [▷▷] ou [◁◁] afin de la positionner juste avant le fichier cherché.

D. — Utilisation d'une imprimante

1. GENERALITES

* L'imprimante n'a qu'une seule fonction :

Imprimer (sur du papier) l'information en provenance de l'ordinateur.

L'information peut être de plusieurs types :

- un programme BASIC ;
- une image écran ;
- un ensemble de données.

* Deux imprimantes Thomson sont utilisables*.

- l'imprimante PR 90.040 permet d'imprimer jusqu'à 40 caractères par ligne ;
- l'imprimante PR 90.080 permet d'imprimer. jusqu'à 80 caractères par ligne

Remarque : les imprimantes à interface Centronics sont également utilisables.

* Vérifier que l'imprimante se trouve bien connectée au TO7 avec les liaisons prévues.

2. IMPRESSION DE PROGRAMME BASIC

Elle se fait au moyen de l'instruction LIST.

Syntaxe :

LIST "LPRT : [(constante)]" [, plage de numéro de ligne].

La constante est le nombre de caractères par ligne : elle est à choisir parmi les entiers positifs inférieurs à 40 ou 80, selon l'imprimante. Si cette constante n'est pas mentionnée le TO7 la considère d'autorité comme égale à 40. L'imprimante liste :

- la partie programme indiquée par plage de numéro de ligne ;
- la totalité du programme si la plage de numéro de ligne n'est pas précisée.

Exemple 1 :

```
LIST "LPRT : (30)", - 150 [ENTREE]
```

permet de lister les 150 premières lignes de programme avec 30 caractères par ligne.

Exemple 2 :

```
LIST "LPRT : (80)" [ENTREE]
```

liste la totalité du programme avec 80 caractères par ligne.

3. IMPRESSION D'UNE IMAGE ECRAN

Elle se fait au moyen de l'instruction SCREENPRINT.

Ainsi ce qui est affiché sur l'écran se retrouve imprimé sur le listing.

Exemple :

```
CLS [ENTREE]          (effacement de l'écran)
SCREENPRINT [ENTREE]
```

SCREENPRINT s'affiche sur l'écran et sur l'imprimante.

4. IMPRESSION D'UN ENSEMBLE DE DONNEES

Elle se fait en trois temps :

a) Ouverture d'un canal en Output vers l'imprimante pour la communication TO7 vers imprimante.

Exemple :

```
OPEN "O", #1, "LPRT : (30)" [ENTREE]
```

Remarques :

. Aucun nom de fichier ne doit être inscrit dans le descripteur de fichier (lorsque c'est l'imprimante qui est concernée).

. Le canal 1 est ouvert en Output vers l'imprimante, les lignes transmises comportent 30 caractères.

b) Impression des données

Elle se fait au moyen des instructions PRINT et PRINT USING (cf chapitre 3).

Exemple :

```
PRINT#1, "BONJOUR MONSIEUR" [ENTREE]
```

Le message BONJOUR MONSIEUR doit s'imprimer sur le listing.

c) Fermeture du canal

Elle se fait au moyen de l'instruction CLOSE.

Exemple :

```
CLOSE 1 [ENTREE]
```

E. — Tableau des codes ASCII

ASCII Code	Character	ASCII Code	Character	ASCII Code	Character
000	NUL	036	$	072	H
001	SOH	037	%	073	I
002	STX	038	&	074	J
003	ETX	039	'	075	K
004	EOT	040	(	076	L
005	ENQ	041	)	077	M
006	ACK	042	*	078	N
007	BEL	043	+	079	O
008	BS	044	,	080	P
009	HT	045	-	081	Q
010	LF	046	.	082	R
011	VT	047	/	083	S
012	FF	048	0	084	T
013	CR	049	1	085	U
014	SO	050	2	086	V
015	SI	051	3	087	W
016	DLE	052	4	088	X
017	DC1	053	5	089	Y
018	DC2	054	6	090	Z
019	DC3	055	7	091	[
020	DC4	056	8	092	\
021	NAK	057	9	093	]
022	SYN	058	:	094	↑
023	ETB	059	;	095	<
024	CAN	060	<	096	`
025	EM	061	=	097	a
026	SUB	062	>	098	b
027	ESCAPE	063	?	099	c
028	FS	064	@	100	d
029	GS	065	A	101	e
030	RS	066	B	102	f
031	US	067	C	103	g
032	SPACE	068	D	104	h
033	!	069	E	105	i
034	"	070	F	106	j
035	#	071	G	107	k

<table>
<tr><td>108</td><td>l</td><td>115</td><td>s</td><td>122</td><td>z</td></tr>
<tr><td>109</td><td>m</td><td>116</td><td>t</td><td>123</td><td>{</td></tr>
<tr><td>110</td><td>n</td><td>117</td><td>u</td><td>124</td><td>|</td></tr>
<tr><td>111</td><td>o</td><td>118</td><td>v</td><td>125</td><td>}</td></tr>
<tr><td>112</td><td>p</td><td>119</td><td>w</td><td>126</td><td>~</td></tr>
<tr><td>113</td><td>q</td><td>120</td><td>x</td><td>127</td><td>DEL</td></tr>
<tr><td>114</td><td>r</td><td>121</td><td>y</td><td></td><td></td></tr>
</table>

ASCII codes are in decimal.
LF=Line Feed, FF=Form Feed, CR=Carriage Return, DEL=Rubout

F. — Analyse du jeu « reverse »

* Les lignes 20 à 60 servent à faire tourner le générateur aléatoire un certain nombre de fois afin d'obtenir des séquences de nombres différentes à chaque début de jeu.

* Les lignes 70 à 110 déterminent la suite initiale.

* Les lignes 130 à 150 permettent au joueur de demander la rotation qu'il veut. Toute rotation supérieure à 9 est refusée. Une rotation nulle signifie l'arrêt du jeu.

* Les lignes 160 à 260 effectuent la rotation demandée.

* Les lignes 220 à 240 permettent de vérifier si la suite obtenue après permutation est 1 2 3 4 5 6 7 8 9. Si ce n'est pas le cas la ligne 230 renvoie à la ligne 130 pour une autre rotation.

* Le sous-programme 1000 gère l'affichage. Le point-virgule après l'instruction PRINT A(I) de la ligne 1020 est indispensable pour obtenir la suite sur une seule ligne.

LA BIBLIOTHEQUE EDIMICRO

Collection " Guides microordinateurs "

de MERLY	GUIDE DE L'APPLE	Tome 1 L'APPLE standard Tome 2 Les extensions Tome 3 Les applications
BAYVEJIEL	GUIDE DE L'ORIC	
BIEBER, PERBOST RENUCCI	GUIDE DU TO 7	

Collection " Jeux "

CHANE-HUNE, DARBOIS	JEUX SUR ORIC
PERBOST, RENUCCI	JEUX SUR TO 7

Collection " Logiciels "

BONNET, DINH	MULTIPLAN SUR APPLE Exercices de Gestion

EDIMICRO
121-127, avenue d'Italie, 75013 PARIS

Imprimé en France. — JOUVE, 18, rue Saint-Denis, 75001 PARIS
N° 11702. Dépôt légal : Septembre 1983